

Monadic Datalog Containment on Trees^{*}

André Frochaux¹, Martin Grohe², and Nicole Schweikardt¹

¹ Goethe-Universität Frankfurt am Main,
{afrochaux,schweika}@informatik.uni-frankfurt.de

² RWTH Aachen University, grohe@informatik.rwth-aachen.de

Abstract. We show that the query containment problem for monadic datalog on finite unranked labeled trees can be solved in 2-fold exponential time when (a) considering unordered trees using the axes *child* and *descendant*, and when (b) considering ordered trees using the axes *firstchild*, *nextsibling*, *child*, and *descendant*. When omitting the *descendant*-axis, we obtain that in both cases the problem is EXPTIME-complete.

1 Introduction

The query containment problem (QCP) is a fundamental problem that has been studied for various query languages. Datalog is a standard tool for expressing queries with recursion. From Cosmadakis et al. [6] and Benedikt et al. [3] it is known that the QCP for *monadic* datalog queries on the class of all finite relational structures is 2EXPTIME-complete. Restricting attention to finite unranked labeled trees, Gottlob and Koch [11] showed that on *ordered* trees the QCP for monadic datalog is EXPTIME-hard and decidable, leaving open the question of a tight bound.

Here we show a matching EXPTIME upper bound for the QCP for monadic datalog on ordered trees using the axes *firstchild*, *nextsibling*, and *child*. When adding the *descendant*-axis, we obtain a 2EXPTIME upper bound. This, in particular, also yields a 2EXPTIME upper bound for the QCP for monadic datalog on *unordered* trees using the axes *child* and *descendant*, and an EXPTIME upper bound for unordered trees using only the *child*-axis. The former result answers a question posed by Abiteboul et al. in [1]. We complement the latter result by a matching lower bound.

The paper is organised as follows. Section 2 fixes the basic notation concerning datalog queries, (unordered and ordered) trees and their representations as logical structures, and summarises basic properties of monadic datalog on trees. Section 3 presents our main results regarding the query containment problem for monadic datalog on trees. Due to space limitations, most technical details had to be deferred to the appendix of this paper.

^{*} This article is the full version of [9].

2 Trees and Monadic Datalog (mDatalog)

Throughout this paper, Σ will always denote a finite non-empty alphabet. By \mathbb{N} we denote the set of non-negative integers, and we let $\mathbb{N}_{\geq 1} := \mathbb{N} \setminus \{0\}$.

Relational Structures. As usual, a *schema* τ consists of a finite number of relation symbols R , each of a fixed *arity* $ar(R) \in \mathbb{N}_{\geq 1}$. A τ -*structure* \mathcal{A} consists of a finite non-empty set A called the *domain* of \mathcal{A} , and a relation $R^{\mathcal{A}} \subseteq A^{ar(R)}$ for each relation symbol $R \in \tau$. It will often be convenient to identify \mathcal{A} with the *set of atomic facts of \mathcal{A}* , i.e., the set $atoms(\mathcal{A})$ consisting of all facts $R(a_1, \dots, a_{ar(R)})$ for all relation symbols $R \in \tau$ and all tuples $(a_1, \dots, a_{ar(R)}) \in R^{\mathcal{A}}$.

If τ is a schema and ℓ is a list of relation symbols, we write τ^ℓ to denote the extension of the schema τ by the relation symbols in ℓ . Furthermore, τ_Σ denotes the extension of τ by new unary relation symbols \mathbf{label}_α , for all $\alpha \in \Sigma$.

Unordered Trees. An *unordered Σ -labeled tree* $T = (V^T, \lambda^T, E^T)$ consists of a finite set V^T of nodes, a function $\lambda^T : V^T \rightarrow \Sigma$ assigning to each node v of T a label $\lambda(v) \in \Sigma$, and a set $E^T \subseteq V^T \times V^T$ of directed edges such that the graph (V^T, E^T) is a rooted tree where edges are directed from the root towards the leaves. We represent such a tree T as a relational structure of domain V^T with unary and binary relations: For each label $\alpha \in \Sigma$, $\mathbf{label}_\alpha(x)$ expresses that x is a node with label α ; $\mathbf{child}(x, y)$ expresses that y is a child of node x ; $\mathbf{root}(x)$ expresses that x is the tree's root node; $\mathbf{leaf}(x)$ expresses that x is a leaf; and $\mathbf{desc}(x, y)$ expresses that y is a descendant of x (i.e., y is a child or a grandchild or ... of x). We denote this relational structure representing T by $\mathcal{S}_u(T)$, but when no confusion arises we simply write T instead of $\mathcal{S}_u(T)$.

The queries we consider for unordered trees are allowed to make use of at least the predicates \mathbf{label}_α and \mathbf{child} . We fix the schema

$$\tau_u := \{\mathbf{child}\}.$$

The representation of unordered Σ -labeled trees as $\tau_{u,\Sigma}$ -structures was considered, e.g., in [1].

Ordered Trees. An *ordered Σ -labeled tree* $T = (V^T, \lambda^T, E^T, order^T)$ has the same components as an unordered Σ -labeled tree and, in addition, $order^T$ fixes for each node u of T , a strict linear order of all the children of u in T .

To represent such a tree as a relational structure, we use the same domain and the same predicates as for unordered Σ -labeled trees, along with three further predicates \mathbf{fc} (“first-child”), \mathbf{ns} (“next-sibling”), and \mathbf{ls} (“last sibling”), where $\mathbf{fc}(x, y)$ expresses that y is the first child of node x (w.r.t. the linear order of the children of x induced by $order^T$); $\mathbf{ns}(x, y)$ expresses that y is the right sibling of x (i.e., x and y have the same parent p , and y is the immediate successor of x in the linear order of p 's children given by $order^T$); and $\mathbf{ls}(x)$ expresses that x is the rightmost sibling (w.r.t. the linear order of the children of x 's parent given by $order^T$). We denote this relational structure representing T by $\mathcal{S}_o(T)$, but when no confusion arises we simply write T instead of $\mathcal{S}_o(T)$.

The queries we consider for ordered trees are allowed to make use of at least the predicates **label**_α, **fc**, and **ns**. We fix the schemas

$$\tau_o := \{\mathbf{fc}, \mathbf{ns}\} \quad \text{and} \quad \tau_{GK} := \tau_o^{\mathbf{root}, \mathbf{leaf}, \mathbf{ls}}.$$

In [11], Gottlob and Koch represented ordered Σ -labeled trees as $\tau_{GK, \Sigma}$ -structures.

Datalog. We assume that the reader is familiar with the syntax and semantics of *datalog* (cf., e.g., [7, 11]). Predicates that occur in the head of some rule of a datalog program \mathcal{P} are called *intensional*, whereas predicates that only occur in the body of rules of \mathcal{P} are called *extensional*. By $\text{idb}(\mathcal{P})$ and $\text{edb}(\mathcal{P})$ we denote the sets of intensional and extensional predicates of \mathcal{P} , resp. We say that \mathcal{P} is of schema τ if $\text{edb}(\mathcal{P}) \subseteq \tau$. We write $\mathcal{T}_{\mathcal{P}}$ to denote the *immediate consequence operator* associated with a datalog program \mathcal{P} . Recall that $\mathcal{T}_{\mathcal{P}}$ maps a set C of atomic facts to the set of all atomic facts that are derivable from C by at most one application of the rules of \mathcal{P} (see e.g. [7, 11]). The monotonicity of $\mathcal{T}_{\mathcal{P}}$ implies that for each finite set C , the iterated application of $\mathcal{T}_{\mathcal{P}}$ to C leads to a fixed point, denoted by $\mathcal{T}_{\mathcal{P}}^{\omega}(C)$, which is reached after a finite number of iterations.

Monadic datalog queries. A datalog program belongs to *monadic datalog* (mDatalog, for short), if all its *intensional* predicates have arity 1.

A *unary monadic datalog query* of schema τ is a tuple $Q = (\mathcal{P}, P)$ where \mathcal{P} is a monadic datalog program of schema τ and P is an intensional predicate of \mathcal{P} . \mathcal{P} and P are called the *program* and the *query predicate* of Q . When evaluated in a finite τ -structure \mathcal{A} that represents a labeled tree T , the query Q results in the unary relation $Q(T) := \{a \in A : P(a) \in \mathcal{T}_{\mathcal{P}}^{\omega}(\text{atoms}(\mathcal{A}))\}$.

The *Boolean monadic datalog query* Q_{Bool} specified by $Q = (\mathcal{P}, P)$ is the Boolean query with $Q_{\text{Bool}}(T) = \mathbf{yes}$ iff the tree's root node belongs to $Q(T)$.

The *size* $\|Q\|$ of a monadic datalog query Q is the length of $Q = (\mathcal{P}, P)$ viewed as a string over a suitable alphabet.

Expressive power of monadic datalog on trees. From Gottlob and Koch [11] we know that on *ordered* Σ -labeled trees represented as $\tau_{GK, \Sigma}$ -structures, monadic datalog can express exactly the same unary queries as monadic second-order logic — for short, we will say “mDatalog(τ_{GK}) = MSO(τ_{GK}) on ordered trees”. Since the **child** and **desc** relations are definable in MSO(τ_{GK}), this implies that mDatalog(τ_{GK}) = mDatalog($\tau_{GK}^{\mathbf{child}, \mathbf{desc}}$) on ordered trees.

On the other hand, using the monotonicity of the immediate consequence operator, one obtains that removing any of the predicates **root**, **leaf**, **ls** from τ_{GK} strictly decreases the expressive power of mDatalog on ordered trees (see [10]). By a similar reasoning one also obtains that on *unordered* trees, represented as $\tau_{u, \Sigma}^{\mathbf{root}, \mathbf{leaf}, \mathbf{desc}}$ -structures, monadic datalog is strictly less expressive than monadic second-order logic, and omitting any of the predicates **root**, **leaf** further reduces the expressiveness of monadic datalog on unordered trees [10].

3 Query Containment for Monadic Datalog on Trees

Let τ_{Σ} be one of the schemas introduced in Section 2 for representing (ordered or unordered) Σ -labeled trees as relational structures. For two unary queries Q_1

and Q_2 of schema τ_Σ we write $Q_1 \subseteq Q_2$ to indicate that for every Σ -labeled tree T we have $Q_1(T) \subseteq Q_2(T)$. Similarly, if Q_1 and Q_2 are *Boolean* queries of schema τ_Σ , we write $Q_1 \subseteq Q_2$ to indicate that for every Σ -labeled tree T , if $Q_1(T) = \mathbf{yes}$ then also $Q_2(T) = \mathbf{yes}$. We write $Q_1 \not\subseteq Q_2$ to indicate that $Q_1 \subseteq Q_2$ does not hold. The *query containment problem* (QCP, for short) is defined as follows:

THE QCP FOR mDatalog(τ) ON TREES
Input: A finite alphabet Σ and
 two (unary or Boolean) mDatalog(τ_Σ)-queries Q_1 and Q_2 .
Question: Is $Q_1 \subseteq Q_2$?

It is not difficult to see that this problem is decidable: the first step is to observe that monadic datalog can effectively be embedded into monadic second-order logic, the second step then applies the well-known result that the monadic second-order theory of finite labeled trees is decidable (cf., e.g., [16,5]).

Regarding ordered trees represented as τ_{GK} -structures, in [11] it was shown that the QCP for unary mDatalog(τ_{GK})-queries on trees is EXPTIME-hard. Our first main result generalises this to unordered trees represented as τ_u -structures:

Theorem 1

The QCP for Boolean mDatalog(τ_u) on unordered trees is EXPTIME-hard.

Our proof proceeds via a reduction from the EXPTIME-complete *two person corridor tiling* (TPCT) problem [4]: For a given instance I of the TPCT-problem we construct (in polynomial time) an alphabet Σ and two Boolean mDatalog($\tau_{u,\Sigma}$)-queries Q_1, Q_2 which enforce that any tree T witnessing that $Q_1 \not\subseteq Q_2$, contains an encoding of a winning strategy for the first player of the TPCT-game associated with I . Using Theorem 1 along with a method of [11] for replacing the **child**-predicate by means of the predicates **fc**, **ns**, we can transfer the hardness result to ordered trees represented by τ_o -structures:

Corollary 2

The QCP for Boolean mDatalog(τ_o) on ordered trees is EXPTIME-hard.

Our second main result provides a matching EXPTIME upper bound for the QCP on ordered trees, even in the presence of all predicates in $\tau_{GK}^{\mathbf{child}}$:

Theorem 3

The QCP for unary mDatalog($\tau_{GK}^{\mathbf{child}}$) on ordered trees belongs to EXPTIME.

Proof (sketch). Consider a schema $\tau \subseteq \tau_{GK}^{\mathbf{child}, \mathbf{desc}}$. By using the automata-theoretic approach [6], a canonical method for deciding the QCP for unary mDatalog(τ) proceeds as follows:

- (1) Transform the input queries Q_1 and Q_2 into *Boolean* queries Q'_1 and Q'_2 on *binary* trees, such that $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$.

- (2) Construct tree automata A_1^{yes} and A_2^{no} such that A_1^{yes} (resp. A_2^{no}) accepts exactly those trees T with $Q'_1(T) = \text{yes}$ (resp. $Q'_2(T) = \text{no}$).
- (3) Construct the product automaton B of A_1^{yes} and A_2^{no} , such that B accepts exactly those trees that are accepted by A_1^{yes} and by A_2^{no} . Afterwards, check if the tree language recognised by B is empty. Note that this is the case if, and only if, $Q_1 \subseteq Q_2$.

Using time polynomial in the size of Q_1 and Q_2 , Step (1) can be achieved in a standard way by appropriately extending the labelling alphabet Σ .

For Step (3), if A_1^{yes} and A_2^{no} are nondeterministic bottom-up tree automata, the construction of B takes time polynomial in the sizes of A_1^{yes} and A_2^{no} , and the emptiness test can be done in time polynomial in the size of B (see e.g. [5]).

The first idea for tackling Step (2) is to use a standard translation of Boolean monadic datalog queries into monadic second-order (MSO) sentences: It is not difficult to see (cf., e.g. [11]) that any Boolean mDatalog(τ)-query Q can be translated in polynomial time into an equivalent MSO-sentence φ_Q of the form

$$\forall X_1 \dots \forall X_n \exists z_1 \dots \exists z_\ell \bigvee_{j=1}^m \gamma_j$$

where n is the number of intensional predicates of Q 's monadic datalog program \mathcal{P} , ℓ and m are linear in the size of Q , and each γ_j is a conjunction of at most b atoms or negated atoms, where b is linear in the maximum number of atoms occurring in the body of a rule of \mathcal{P} . Applying the standard method for translating MSO-sentences into tree automata (cf., e.g., [16]), we can translate the sentence $\neg\varphi_Q$ into a nondeterministic bottom-up tree-automaton A^{no} that accepts a tree T iff $Q(T) = \text{no}$. This automaton has $2^{(m' \cdot c^{b'})}$ states, where m' and b' are linear in m and b , resp., and c is a constant not depending on Q or Σ ; and A^{no} can be constructed in time polynomial in $|\Sigma| \cdot 2^{n+\ell+m' \cdot c^{b'}}$.

Using the subset construction, one obtains an automaton A^{yes} which accepts a tree T iff $Q(T) = \text{yes}$; and this automaton has $2^{2^{(m' \cdot c^{b'})}}$ states.

Note that, a priori, b' might be linearly related to the size of Q . Thus, the approach described so far leads to a *3-fold exponential* algorithm that solves the QCP for unary mDatalog(τ)-queries.

In case that τ does *not* contain the **desc**-predicate, we obtain a 2-fold exponential algorithm as follows: At the end of Step (1) we rewrite Q'_1 and Q'_2 into queries that do not contain the **child**-predicate, and we transform both queries into *tree marking normal form* (TMNF), i.e., a normal form in which bodies of rules consist of at most two atoms, at least one of which is unary. From [11] we obtain that these transformations can be done in time polynomial in the size of Q'_1 and Q'_2 . Note that for TMNF-queries, the parameters b and b' are constant (i.e., they do not depend on the query), and thus the above description shows that for TMNF-queries the automaton A_2^{no} can be constructed in 1-fold exponential time, and A_1^{yes} can be constructed in 2-fold exponential time.

Finally, the key idea to obtain a *1-fold exponential* algorithm solving the QCP is to use a different construction for the automaton A_1^{yes} , which does not use the detour via an MSO-formula but, instead, takes a detour via a *two-way*

alternating tree automaton (2ATA): We show that a Boolean TMNF-query can be translated, in polynomial time, into a 2ATA \hat{A}_1^{yes} that accepts a tree T iff $Q_1(T) = \text{yes}$. It is known that, within 1-fold exponential time, a 2ATA can be transformed into an equivalent nondeterministic bottom-up tree automaton (this was claimed already in [6]; detailed proofs of more general results can be found in [17,14]). In summary, this leads to a 1-fold exponential algorithm for solving the QCP for $\text{mDatalog}(\tau_{GK}^{\text{child}})$ on ordered trees. \square

Since $\tau_u^{\text{root,leaf}} \subseteq \tau_{GK}^{\text{child}}$, Theorem 3 immediately implies:

Corollary 4 *The QCP for unary $\text{mDatalog}(\tau_u^{\text{root,leaf}})$ on unordered trees belongs to EXPTIME.*

It remains open if the EXPTIME-membership results of Theorem 3 and Corollary 4 can be generalised to queries that also use the descendant predicate **desc**. However, the first approach described in the proof of Theorem 3 yields a 3-fold exponential algorithm. We can improve this by using methods and results from [11] and [12] to eliminate the **desc**-predicate at the expense of an exponential blow-up of the query size. Afterwards, we apply the algorithms provided by Theorem 3 and Corollary 4. This leads to the following:

Theorem 5 *The QCP for unary $\text{mDatalog}(\tau_u^{\text{root,leaf,desc}})$ on unordered trees and for unary $\text{mDatalog}(\tau_{GK}^{\text{child,desc}})$ on ordered trees can be solved in 2-fold exponential time.*

Open Question. It remains open to close the gap between the EXPTIME lower and the 2EXPTIME upper bound for the case where the *descendant*-axis is involved.

Acknowledgment. The first author would like to thank Mariano Zelke for countless inspiring discussions and helpful hints on and off the topic.

References

1. S. Abiteboul, P. Bourhis, A. Muscholl, and Z. Wu. Recursive queries on trees and data trees. In *Proc. ICDT'13*, pages 93–104, 2013.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
3. M. Benedikt, P. Bourhis, and P. Senellart. Monadic datalog containment. In *Proc. ICALP'12*, pages 79–91, 2012.
4. B. S. Chlebus. Domino-tiling games. *J. Comput. Syst. Sci.*, 32(3):374–392, 1986.
5. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at <http://www.grappa.univ-lille3.fr/tata>, 2008. release November, 18th 2008.
6. S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. Decidable optimization problems for database logic programs. In *Proc. STOC'88*, pages 477–490, 1988.
7. E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
8. J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2005.

9. A. Frochaux, M. Grohe, and N. Schweikardt. Monadic datalog containment on trees. In *Proceedings of the 8th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2014), Cartagena, Colombia, June 2-6, 2014*, CEUR Workshop Proceedings. CEUR-WS.org, 2014.
10. A. Frochaux and N. Schweikardt. A note on monadic datalog on unranked trees. *Technical Report, available at CoRR*, abs/1310.1316, 2013.
11. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.
12. G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. *J. ACM*, 53(2):238–272, 2006.
13. C. Löding. Basics on tree automata. In D. D’Souza and P. Shankar, editors, *Modern Applications of Automata Theory*. World Scientific, 2012.
14. S. Maneth, S. Friesen, and H. Seidl. Type-Checking Tree Walking Transducers. In D. D’Souza and P. Shankar, editors, *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2010.
15. F. Neven. Automata, Logic, and XML. In *Proc. CSL’02*, pages 2–26, 2002.
16. W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages*, volume 3, pages 389–455. Springer-Verlag, 1997.
17. M. Vardi. Reasoning about the past with two-way automata. In *Proc. ICALP’98*, pages 628–641, 1998.

APPENDIX

This appendix contains technical details which were omitted in the main part of the paper.

- Appendix A contains further basic notation, including a precise definition of the syntax and semantics of datalog.
- Appendix B gives a detailed proof of Theorem 1.
- Appendix C provides a proof of Corollary 2.
- Appendix D gives a detailed proof of Theorem 3.
- Appendix E presents a proof of Theorem 5.

A Basic Notation and Syntax and Semantics of Datalog

Basic notation For a set S we write 2^S to denote the power set of S .

Let τ be a schema suitable for representing ordered (or unordered) Σ -labeled trees. Two mDatalog(τ)-queries Q and Q' are called *equivalent* if $Q(T) = Q'(T)$ is true for all finite ordered (or unordered, resp.) Σ -labeled trees T .

The following definition of datalog is basically taken from [7].

Syntax of datalog A *datalog rule* is an expression of the form $h \leftarrow b_1, \dots, b_n$, for $n \in \mathbb{N}$, where h, b_1, \dots, b_n are called *atoms* of the rule, h is called the rule's *head*, and b_1, \dots, b_n (understood as a conjunction of atoms) is called the *body*. Each atom is of the form $P(x_1, \dots, x_m)$ where P is a predicate of some arity $m \in \mathbb{N}_{\geq 1}$ and x_1, \dots, x_m are variables. Rules are required to be *safe* in the sense that all variables appearing in the head also have to appear in the body.

A *datalog program* is a finite set of datalog rules. Let \mathcal{P} be a datalog program and let r be a datalog rule. We write $\text{var}(r)$ for the set of all variables occurring in the rule r , and we let $\text{var}(\mathcal{P}) := \bigcup_{r \in \mathcal{P}} \text{var}(r)$. Predicates that occur in the head of some rule of \mathcal{P} are called *intensional*, whereas predicates that only occur in the body of rules of \mathcal{P} are called *extensional*. We write $\text{idb}(\mathcal{P})$ and $\text{edb}(\mathcal{P})$ to denote the sets of intensional and extensional predicates of \mathcal{P} , respectively. We say that \mathcal{P} is of schema τ if $\text{edb}(\mathcal{P}) \subseteq \tau$.

Semantics of datalog For defining the semantics of datalog, let τ be a schema, let \mathcal{P} be a datalog program of schema τ , let A be a domain, and let

$$F_{\mathcal{P},A} := \{ R(a_1, \dots, a_r) : R \in \tau \cup \text{idb}(\mathcal{P}), r = \text{ar}(R), a_1, \dots, a_r \in A \}$$

be the set of all *atomic facts over A*. A *valuation* β for \mathcal{P} in A is a function $\beta : (\text{var}(\mathcal{P}) \cup A) \rightarrow A$ where $\beta(a) = a$ for all $a \in A$. For an atom $b := P(x_1, \dots, x_m)$ occurring in a rule of \mathcal{P} we let $\beta(b) := P(\beta(x_1), \dots, \beta(x_m))$. The *immediate consequence operator* $\mathcal{T}_{\mathcal{P}}$ induced by \mathcal{P} on A maps every $C \subseteq F_{\mathcal{P},A}$ to

$$\mathcal{T}_{\mathcal{P}}(C) := C \cup \left\{ \beta(h) : \begin{array}{l} \text{there is a rule } h \leftarrow b_1, \dots, b_n \text{ in } \mathcal{P} \text{ and a valuation } \\ \beta \text{ for } \mathcal{P} \text{ in } A \text{ such that } \beta(b_1), \dots, \beta(b_n) \in C \end{array} \right\}.$$

Clearly, $\mathcal{T}_{\mathcal{P}}$ is *monotone*, i.e., $\mathcal{T}_{\mathcal{P}}(C) \subseteq \mathcal{T}_{\mathcal{P}}(D)$ holds for all $C \subseteq D \subseteq F_{\mathcal{P},A}$. Letting $\mathcal{T}_{\mathcal{P}}^0(C) := C$ and $\mathcal{T}_{\mathcal{P}}^{i+1}(C) := \mathcal{T}_{\mathcal{P}}(\mathcal{T}_{\mathcal{P}}^i(C))$ for all $i \in \mathbb{N}$, one obtains

$$C = \mathcal{T}_{\mathcal{P}}^0(C) \subseteq \mathcal{T}_{\mathcal{P}}^1(C) \subseteq \dots \subseteq \mathcal{T}_{\mathcal{P}}^i(C) \subseteq \mathcal{T}_{\mathcal{P}}^{i+1}(C) \subseteq \dots \subseteq F_{\mathcal{P},A}.$$

For a finite domain A , the set $F_{\mathcal{P},A}$ is finite, and hence there is an $i_0 \in \mathbb{N}$ such that $\mathcal{T}_{\mathcal{P}}^{i_0}(C) = \mathcal{T}_{\mathcal{P}}^i(C)$ for all $i \geq i_0$. In particular, the set $\mathcal{T}_{\mathcal{P}}^\omega(C) := \mathcal{T}_{\mathcal{P}}^{i_0}(C)$ is a *fixpoint* of the operator $\mathcal{T}_{\mathcal{P}}$. By the theorem of Knaster and Tarski we know that this fixpoint is the *smallest* fixpoint of $\mathcal{T}_{\mathcal{P}}$ which contains C .

B EXPTIME-Hardness: Proof of Theorem 1

The aim of this appendix is to prove the following:

Theorem 1 (restated)

The QCP for Boolean mDatalog(τ_u) on unordered trees is EXPTIME-hard.

We will show this by first proving the according hardness result for the schema $\tau_u^{\text{root,leaf}}$. Afterwards, we will construct a polynomial-time reduction which provides the same hardness result also for the schema τ_u .

B.1 EXPTIME-hardness result for the schema $\tau_u^{\text{root,leaf}}$

This subsection's main result is

Proposition 6

The QCP for Boolean mDatalog($\tau_u^{\text{root,leaf}}$) on unordered trees is EXPTIME-hard.

Proof. Our proof proceeds by reduction from the EXPTIME-complete *two person corridor tiling* problem (TPCT) [4]. The task of the TPCT-problem is to decide whether the first player in the following *two person corridor tiling game* has a winning strategy.

There are two players: Player 1 (the *Constructor*) and Player 2 (the *Saboteur*). The game board is a corridor of a given width n and an unbounded length. There is a finite set \mathcal{D} of types of *tiles* (or, *dominoes*), and from every tile type, an unlimited number of tiles is available. The first row f (of width n) of tiles, as well as the designated last row ℓ (of width n) of tiles are given.

The players alternately select a tile and put it into the next vacant position (row-wise from left to right); Player 1 starts at the leftmost position of the second row. Both players have to respect horizontal and vertical constraints, given by two sets $H, V \subseteq \mathcal{D}^2$. A tile d chosen for the j -th column of the i -th row has to fit to its vertical neighbour d_v in the j -th column of the $(i-1)$ -th row in the sense that $(d_v, d) \in V$. Furthermore, if $j \geq 2$, then tile d also has to fit to its horizontal neighbour d_h in the $(j-1)$ -th column of the i -th row in the sense that $(d_h, d) \in H$. If a player is unable to choose a fitting tile, Player 1 loses and the game ends.

The ultimate goal of Player 1 is to produce a tiling whose last row is ℓ ; in this case he wins and the game ends. Player 2 wins if either the game goes on for an infinite number of steps, or one of the players gets stuck in a situation where he cannot find a fitting tile.

The *two person corridor tiling problem* (TPCT) is the following decision problem.

TPCT

Input: A tuple $I = (\mathcal{D}, H, V, n, f, \ell)$ such that \mathcal{D} is a finite set, $H, V \subseteq \mathcal{D}^2$, $n \geq 2$, $f, \ell \in \mathcal{D}^n$.

Question: Does Player 1 have a winning strategy in the two person corridor tiling game specified by I ?

Theorem 7 (Chlebus [4]) *The problem TPCT is EXPTIME-complete.*

Note that EXPTIME is closed under complementation. Thus, for proving Proposition 6 it suffices to give a polynomial-time reduction from TPCT to the *complement* of the QCP for mDatalog($\tau_u^{\text{root}, \text{leaf}}$) on unordered trees. For a given TPCT-instance $I = (\mathcal{D}, H, V, n, f, \ell)$ we will construct a finite alphabet Σ and two Boolean mDatalog($\tau_{u, \Sigma}^{\text{root}, \text{leaf}}$)-queries Q_1, Q_2 , such that

$$\begin{aligned} & \text{Player 1 has a winning strategy in the} \\ & \text{two person corridor tiling game specified by } I \\ \iff & \text{ there exists an unordered } \Sigma\text{-labeled tree } T \text{ such that} \\ & Q_1(T) = \mathbf{yes} \text{ and } Q_2(T) = \mathbf{no} \text{ (i.e., } Q_1 \not\subseteq Q_2\text{).} \end{aligned}$$

We will represent strategies for Player 1 by Σ -labeled trees. The query Q_1 will describe “necessary properties” which are met by every tree that describes a winning strategy for Player 1, but also by some other trees. The query Q_2 will describe certain “forbidden properties” such that a tree which has these properties for sure does *not* describe a winning strategy for Player 1.

The following representation of a winning strategy for Player 1 is basically taken from [13]. We represent a strategy for Player 1 by an unordered Σ -labeled tree with

$$\Sigma := \mathcal{D} \times \{1, 2, \perp, !\}.$$

The first component of a letter $(d, i) \in \Sigma$ corresponds to the tile d that has been played, while the second component indicates whose turn it is to place the next tile (1 for Player 1, 2 for Player 2, \perp in case the game is over because a vertical or horizontal constraint was violated, and $!$ in case that the game is over because Player 1 has won). In the following, we will say that a node is labeled d (for some $d \in \mathcal{D}$) to express that its label belongs to $\{d\} \times \{1, 2, \perp, !\}$. Accordingly, we will say that a node is labeled i (for some $i \in \{1, 2, \perp, !\}$) to express that its label belongs to $\mathcal{D} \times \{i\}$.

A finite Σ -labeled tree T is called *good* if it satisfies the following conditions (1)–(9). It is not difficult to verify that Player 1 has a winning strategy if, and only if, there exists a finite Σ -labeled tree that is good.

- (1) The root is labeled by $(d, 2)$ for some $d \in \mathcal{D}$. (This indicates that at the beginning of the game, Player 1 chooses tile d , and Player 2 is the one to play in the next step).
- (2) Nodes with labels \perp or $!$ are leaves.
- (3) Nodes with labels in $\mathcal{D} \times \{1\}$ have at least one child. (Such a child describes the choice made by Player 1 in the next step).
- (4) Nodes with labels in $\mathcal{D} \times \{2\}$ have at least $|\mathcal{D}|$ children — one for each tile type $d \in \mathcal{D}$. (These children represent the potential choices that Player 2 might make in the next step).
- (5) There is no node labeled 1 or 2 such that all of its children are labeled by \perp . (I.e., the game never gets stuck).

- (6) Labels from $\mathcal{D} \times \{1\}$ and $\mathcal{D} \times \{2\}$ alternate on each path from the root to a leaf. (I.e., both players alternately choose a tile).
- (7) If a node x is labeled $!$, then the number of nodes visited by the path from the root to x is a multiple of n and the last n nodes on this path are labeled according to ℓ . (This means that the last n nodes of the path describe a row which has the desired labeling ℓ .)
- (8) At each node x labeled (d, i) with $i \neq \perp$, the tile d respects the horizontal and the vertical constraints.
- (9) At each node labeled (d, \perp) for some $d \in \mathcal{D}$, the tile d violates the horizontal or the vertical constraints.

To be precise, the conditions (8) and (9) mean the following. We define the *depth* of a node as follows: The root has depth 1; and for each node x of depth j , all children of x are of depth $j+1$.

- (a) A node x labeled with tile $d \in \mathcal{D}$ *respects the horizontal constraints* if x
 - is either of depth congruent 1 modulo n (and thus corresponds to a position in the 1-st column of a row),
 - or we have $(d_h, d) \in H$, where the parent of x is labeled with tile $d_h \in \mathcal{D}$ (i.e., x corresponds to a position where tile d is chosen in some column $j \geq 2$, and this tile fits to its horizontal neighbour d_h in column $j-1$).
- (b) A node x labeled with a tile $d \in \mathcal{D}$ *respects the vertical constraints* if x
 - is either is of depth $j \in \{1, \dots, n\}$ and we have $(f_j, d) \in H$ (i.e., x corresponds to the j -th position in the second row and fits to the j -th entry f_j of the first row f),
 - or it is of depth $j \geq n+1$ and we have $(d_v, d) \in V$, where the ancestor of x at depth $j-n$ is labeled with tile $d_v \in \mathcal{D}$ (i.e., x corresponds to a position where tile d is chosen in some row $i \geq 3$, and this tile fits to its vertical neighbour d_v in row $i-1$).

As noted above, Player 1 has a winning strategy if, and only if, there exists a finite Σ -labeled tree T that is good, i.e., that satisfies the conditions (1)–(9). The first idea towards completing the proof of Proposition 6 is to try to find monadic Datalog queries Q_1 and Q_2 such that for any Σ -labeled tree T the following is true: T is good if, and only if, $Q_1(T) = \mathbf{yes}$ and $Q_2(T) = \mathbf{no}$. In fact, it is not difficult to construct for each condition (c) with $c \neq 4$ and $c \neq 5$ a Boolean mDatalog($\tau_u^{\text{root}, \text{leaf}}$)-query Q^c such that for any Σ -labeled tree T we have:

$$Q^c(T) = \mathbf{yes} \iff T \text{ violates condition } (c).$$

However, for the conditions (4) and (5), we were unable to find according monadic datalog queries which precisely characterise all trees that violate (or all trees that fulfill) these conditions.

As a remedy, we define a notion of *almost-good* trees in such a way that the following is true:

- (i) Every *almost-good* tree T contains a *good* tree;
and every *good* tree also is *almost-good*.

- (ii) We can find Boolean mDatalog($\tau_{u,\Sigma}^{\text{root,leaf}}$)-queries Q_1, Q_2 such that for any Σ -labeled tree T the following is true: T is *almost-good* if, and only if, $Q_1(T) = \text{yes}$ and $Q_2(T) = \text{no}$.

For defining the notion of *almost-good* trees, we need the following notation. Let T be an unordered Σ -labeled tree. By performing a bottom-up scan of T , we define the set of nodes that are *candidates* as follows:

- Every *leaf* of T that is labeled \perp or $!$ is a *candidate*.
- For each node x of T that is labeled 1, x is a *candidate* if x has a child that is a *candidate* and that is *not* labeled \perp .
- For each node x of T that is labeled 2, x is a *candidate* if
 - for each $d \in \mathcal{D}$, x has a child that is a *candidate* and that is labeled d ,
 - and x has child that is a *candidate* and that is *not* labeled \perp .

Now, we perform a top-down scan of T to define the set of nodes that are *relevant* as follows:

- The root of T is *relevant* if it is labeled in $\mathcal{D} \times \{2\}$ and it is a *candidate*.
- For each non-root node x of T , x is *relevant* if it is a *candidate* and its parent is *relevant*.

Note that according to this definition, in particular, the following is true:

- Every *relevant* node of T either is a leaf of T or has a child that is *relevant*.
- If the root of T is *relevant*, then it is labeled in $\mathcal{D} \times \{2\}$, and the set of all *relevant* nodes of T forms a tree, which we will call T_{Relevant} .
- Relevant nodes with labels \perp or $!$ are leaves.
- Every *relevant* node with label in $\mathcal{D} \times \{1\}$ has a *relevant* child that is *not* labeled \perp .
- Every *relevant* node with label in $\mathcal{D} \times \{2\}$ has, for each $d \in \mathcal{D}$, a *relevant* child labeled d ; and it has a *relevant* child that is *not* labeled \perp .

Thus, the following is true for every Σ -labeled tree T :

- (*) : If the root of T is *relevant*, then
the tree T_{Relevant} satisfies the conditions (1)–(5).

Furthermore, note that if T is *good*, then $T_{\text{Relevant}} = T$.

We say that a Σ -labeled tree T is *almost-good* if its root node is *relevant* and the tree T_{Relevant} is *good*, i.e., satisfies the conditions (6)–(9).

Our next goal is to construct an mDatalog($\tau_{b,\Sigma}^{\text{root,leaf}}$)-program $\mathcal{P}_{\text{Relevant}}$ which constructs, in an intensional predicate called *Relevant*, the set of all *relevant* nodes. We start with $\mathcal{P}_{\text{Relevant}} := \emptyset$. To access the parts d and i of a node-label $(d, i) \in \Sigma$, it will be convenient to include into $\mathcal{P}_{\text{Relevant}}$ the rules

$$\text{label}_d(x) \leftarrow \text{label}_{(d,i)}(x) \quad \text{and} \quad \text{label}_i(x) \leftarrow \text{label}_{(d,i)}(x)$$

for every letter $(d, i) \in \Sigma$. Furthermore, for all $d, d' \in \mathcal{D}$ and $i, i' \in \{1, 2, \perp, !\}$ with $d \neq d'$ and $i \neq i'$ we add to $\mathcal{P}_{\text{Relevant}}$ the rules

$$\text{label}_{\neq d}(x) \leftarrow \text{label}_{d'}(x) \quad \text{and} \quad \text{label}_{\neq i}(x) \leftarrow \text{label}_{i'}(x).$$

To describe the *candidate* nodes, we add to $\mathcal{P}_{\text{Relevant}}$ the rules

$$\begin{aligned} \text{Candidate}(x) &\leftarrow \text{leaf}(x), \text{label}_\perp(x) \\ \text{Candidate}(x) &\leftarrow \text{leaf}(x), \text{label}_1(x) \\ \text{Candidate}(x) &\leftarrow \text{label}_1(x), \text{child}(x, y), \text{Candidate}(y), \text{label}_{\neq \perp}(y), \end{aligned}$$

as well as the following rule, where d_1, \dots, d_m is a list of all elements in \mathcal{D} :

$$\begin{aligned} \text{Candidate}(x) &\leftarrow \text{label}_2(x), \text{child}(x, y_1), \dots, \text{child}(x, y_m), \\ &\quad \text{Candidate}(y_1), \dots, \text{Candidate}(y_m), \\ &\quad \text{label}_{d_1}(y_1), \dots, \text{label}_{d_m}(y_m), \\ &\quad \text{child}(x, y), \text{Candidate}(y), \text{label}_{\neq \perp}(y). \end{aligned}$$

To describe the *relevant* nodes, we add to $\mathcal{P}_{\text{Relevant}}$ the rules

$$\begin{aligned} \text{Relevant}(x) &\leftarrow \text{root}(x), \text{Candidate}(x), \text{label}_2(x) \\ \text{Relevant}(x) &\leftarrow \text{Candidate}(x), \text{child}(y, x), \text{Relevant}(y) \end{aligned}$$

This completes the definition of the monadic datalog program $\mathcal{P}_{\text{Relevant}}$. Obviously, the following is true:

Claim 1 $\mathcal{P}_{\text{Relevant}}$ can be constructed in time polynomial in the size of Σ . Furthermore, for the unary query $Q_{\text{Relevant}} := (\mathcal{P}_{\text{Relevant}}, \text{Relevant})$ the following is true: For every unordered Σ -labeled tree T , the set $Q_{\text{Relevant}}(T)$ contains exactly those nodes of T that are relevant.

Recall that our overall goal is to find Boolean queries Q_1 and Q_2 that satisfy condition (ii). We choose Q_1 to be the query that returns “**yes**” exactly for those trees T whose root is *relevant*. I.e., the program of Q_1 is obtained from $\mathcal{P}_{\text{Relevant}}$ by adding the rule

$$\text{accept}(x) \leftarrow \text{root}(x), \text{Relevant}(x)$$

and the query predicate of Q_1 is the predicate **accept**. From (*) we know that the following is true:

Claim 2 Q_1 can be constructed in time polynomial in the size of Σ ; and for every Σ -labeled tree T we have $Q_1(T) = \text{yes}$ if, and only if, the root of T is relevant and the tree T_{Relevant} satisfies the conditions (1)–(5).

Our next goal is to construct a Boolean query Q_2 that returns “**yes**” exactly for those trees T where the tree T_{Relevant} violates one of the conditions (6)–(9). Once we have achieved this, we know that for any tree T the following is true: $Q_1(T) = \text{yes}$ and $Q_2(T) = \text{no}$ if, and only if, the tree T_{Relevant} satisfies the conditions (1)–(9), and hence witnesses that Player 1 has a winning strategy for the two person corridor tiling game specified by $I = (\mathcal{D}, H, V, n, f, \ell)$.

To construct Q_2 , we start with the monadic Datalog program $\mathcal{P}_2 := \mathcal{P}_{\text{Relevant}}$ and successively add rules to \mathcal{P}_2 .

To detect a violation of condition (6), we add to \mathcal{P}_2 the rules

$$\begin{aligned} \mathbf{reject}^{(6)}(z) &\leftarrow \mathbf{Relevant}(x), \mathbf{Relevant}(y), \\ &\quad \mathbf{child}(x, y), \mathbf{label}_1(x), \mathbf{label}_1(y), \mathbf{root}(z) \\ \mathbf{reject}^{(6)}(z) &\leftarrow \mathbf{Relevant}(x), \mathbf{Relevant}(y), \\ &\quad \mathbf{child}(x, y), \mathbf{label}_2(x), \mathbf{label}_2(y), \mathbf{root}(z). \end{aligned}$$

This way, $T_{\mathbf{Relevant}}$ violates condition (6) if, and only if, the root of T gets assigned the predicate $\mathbf{reject}^{(6)}$. Thus, the Boolean query specified by $(\mathcal{P}_2, \mathbf{reject}^{(6)})$ returns “yes” for exactly those trees T where $T_{\mathbf{Relevant}}$ violates condition (6).

To detect a violation of the conditions (7)–(9), it will be convenient to use predicates $Column_j$ for each $j \in \{1, \dots, n\}$, such that $Column_j(x)$ indicates that node x corresponds to a tile placed in column j of the corridor. Thus, we add to \mathcal{P}_2 the rules

$$\begin{aligned} Column_1(x) &\leftarrow \mathbf{root}(x), \mathbf{Relevant}(x) \\ Column_1(x) &\leftarrow \mathbf{child}(y, x), Column_n(y), \mathbf{Relevant}(y), \mathbf{Relevant}(x) \end{aligned}$$

and for each $j \in \{2, \dots, n\}$ the rule

$$Column_j(x) \leftarrow \mathbf{child}(y, x), Column_{j-1}(y), \mathbf{Relevant}(y), \mathbf{Relevant}(x).$$

Furthermore, for each $j \in \{1, \dots, n-1\}$ we add to \mathcal{P}_2 the rule

$$Column_{\neq n}(x) \leftarrow Column_j(x)$$

and for each $j \in \{2, \dots, n\}$ we add to \mathcal{P}_2 the rule

$$Column_{\neq 1}(x) \leftarrow Column_j(x).$$

To detect a violation of condition (7), we add to \mathcal{P}_2 the rule

$$\mathbf{reject}^{(7)}(z) \leftarrow \mathbf{label}_1(x), Column_{\neq n}(x), \mathbf{Relevant}(x), \mathbf{root}(z)$$

and for each $j \in \{1, \dots, n\}$ we add the rule

$$\begin{aligned} \mathbf{reject}^{(7)}(z) &\leftarrow \mathbf{label}_1(x_n), \mathbf{child}(x_1, x_2), \dots, \mathbf{child}(x_{n-1}, x_n), \\ &\quad \mathbf{label}_{\neq \ell_j}(x_j), \mathbf{Relevant}(x_1), \dots, \mathbf{Relevant}(x_n), \mathbf{root}(z) \end{aligned}$$

where ℓ_j denotes the j -th position of the designated last row ℓ .

This way, $T_{\mathbf{Relevant}}$ violates condition (7) if, and only if, the root of T gets assigned the predicate $\mathbf{reject}^{(7)}$. Hence, the Boolean query specified by $(\mathcal{P}_2, \mathbf{reject}^{(7)})$ returns “yes” for exactly those trees T where $T_{\mathbf{Relevant}}$ violates condition (7). Note that \mathcal{P}_2 can be constructed in time polynomial in the size of Σ and n .

To detect a violation of condition (8), it will be convenient to use predicates $Buggy_H$ and $Buggy_V$, such that $Buggy_H(x)$ (resp., $Buggy_V(x)$) indicates that

node x violates the horizontal (resp., the vertical) constraints. Thus, for all $(d_h, d) \in \mathcal{D}^2 \setminus H$, we add to \mathcal{P}_2 the rule

$$\begin{aligned} Buggy_H(x) \leftarrow & Column_{\neq 1}(x), \mathbf{child}(y, x), \mathbf{label}_{d_h}(y), \mathbf{label}_d(x), \\ & Relevant(y), Relevant(x). \end{aligned}$$

Similarly, for all $(d_v, d) \in \mathcal{D}^2 \setminus V$, we add to \mathcal{P}_2 the rule

$$\begin{aligned} Buggy_V(x) \leftarrow & \mathbf{child}(y_1, y_2), \dots, \mathbf{child}(y_{n-1}, y_n), \mathbf{child}(y_n, x), \\ & \mathbf{label}_{d_v}(y_1), \mathbf{label}_d(x), \\ & Relevant(y_1), \dots, Relevant(y_n), Relevant(x). \end{aligned}$$

To detect nodes that correspond to tiles placed in the corridor's second row, i.e., tiles that must fit to the given first row $f = (f_1, \dots, f_n) \in \mathcal{D}^n$, we furthermore add for each $j \in \{1, \dots, n\}$ and each $d \in \mathcal{D}$ with $(f_j, d) \notin V$, the rule

$$\begin{aligned} Buggy_V(x_j) \leftarrow & \mathbf{root}(x_1), \mathbf{child}(x_1, x_2), \dots, \mathbf{child}(x_{n-1}, x_n), \\ & \mathbf{label}_d(x_j), Relevant(x_1), \dots, Relevant(x_n) \end{aligned}$$

To detect a violation of condition (8) we add to \mathcal{P}_2 the rules

$$\begin{aligned} \mathbf{reject}^{(8)}(z) \leftarrow & \mathbf{label}_{\neq \perp}(x), Buggy_H(x), Relevant(x), \mathbf{root}(z) \\ \mathbf{reject}^{(8)}(z) \leftarrow & \mathbf{label}_{\neq \perp}(x), Buggy_V(x), Relevant(x), \mathbf{root}(z). \end{aligned}$$

This way, $T_{Relevant}$ violates condition (8) if, and only if, the root of T gets assigned the predicate $\mathbf{reject}^{(8)}$. Hence, the Boolean query specified by $(\mathcal{P}_2, \mathbf{reject}^{(8)})$ returns “yes” for exactly those trees T where $T_{Relevant}$ violates condition (8). Note that \mathcal{P}_2 can be constructed in time polynomial in the size of Σ, n, \mathcal{D} .

To detect a violation of condition (9), it will be convenient to use predicates $Okay_H$ and $Okay_V$, such that $Okay_H(x)$ (resp., $Okay_V(x)$) indicates that node x satisfies the horizontal (resp., the vertical) constraints. Thus, for all $(d_h, d) \in H$, we add to \mathcal{P}_2 the rules

$$\begin{aligned} Okay_H(x) \leftarrow & Column_1(x) \\ Okay_H(x) \leftarrow & Column_{\neq 1}(x), \mathbf{child}(y, x), \mathbf{label}_{d_h}(y), \mathbf{label}_d(x). \end{aligned}$$

Similarly, for all $(d_v, d) \in V$, we add to \mathcal{P}_2 the rules

$$\begin{aligned} Okay_V(x) \leftarrow & \mathbf{child}(y_1, y_2), \dots, \mathbf{child}(y_{n-1}, y_n), \mathbf{child}(y_n, x), \\ & \mathbf{label}_{d_v}(y_1), \mathbf{label}_d(x). \end{aligned}$$

To detect nodes that correspond to tiles placed in the corridor's second row, i.e., tiles that must fit to the given first row $f = (f_1, \dots, f_n) \in \mathcal{D}^n$, we furthermore add for each $j \in \{1, \dots, n\}$ and each $d \in \mathcal{D}$ with $(f_j, d) \in V$, the rule

$$Okay_V(x_j) \leftarrow \mathbf{root}(x_1), \mathbf{child}(x_1, x_2), \dots, \mathbf{child}(x_{n-1}, x_n), \mathbf{label}_d(x_j).$$

To detect a violation of condition (9) we add to \mathcal{P}_2 the rule

$$\mathbf{reject}^{(9)}(z) \leftarrow \mathbf{label}_\perp(x), \mathit{Okay}_H(x), \mathit{Okay}_V(x), \mathit{Relevant}(x), \mathbf{root}(z).$$

This way, T_{Relevant} violates condition (9) if, and only if, the root of T gets assigned the predicate $\mathbf{reject}^{(9)}$. Hence, the Boolean query specified by $(\mathcal{P}_2, \mathbf{reject}^{(9)})$ returns “yes” for exactly those trees T where T_{Relevant} violates condition (9). Note that \mathcal{P}_2 can be constructed in time polynomial in the size of $\Sigma, n, \mathcal{D}, H, V$.

Finally, for each $c \in \{6, 7, 8, 9\}$ we add to \mathcal{P}_2 the rule

$$\mathbf{reject}(z) \leftarrow \mathbf{reject}^{(c)}(z)$$

and we let Q_2 be the Boolean monadic datalog query specified by $(\mathcal{P}_2, \mathbf{reject})$. By our construction, the following holds:

Claim 3 Q_2 can be constructed in time polynomial in the size of $\Sigma, n, \mathcal{D}, H, V$; and for every Σ -labeled tree T we have $Q_2(T) = \mathbf{yes}$ if, and only if, the tree T_{Relevant} violates one of the conditions (6)–(9).

In summary, for each TPCT-instance $I = (\mathcal{D}, H, V, n, f, \ell)$, we can construct within polynomial time the alphabet $\Sigma := \mathcal{D} \times \{1, 2, \perp, !\}$ and two Boolean mDatalog($\tau_{b, \Sigma}^{\mathbf{root}, \mathbf{leaf}}$)-queries Q_1, Q_2 such that the following is true for every unordered Σ -labeled tree T :

$$\begin{aligned} Q_1(T) = \mathbf{yes} \text{ and } Q_2(T) = \mathbf{no} \\ \iff \begin{aligned} &\text{the root of } T \text{ is } \mathit{relevant} \text{ and} \\ &\text{the tree } T_{\mathit{Relevant}} \text{ satisfies the conditions (1)–(9).} \end{aligned} \end{aligned}$$

Thus, $Q_1 \not\subseteq Q_2$ if, and only if, Player 1 has a winning strategy in the two person corridor tiling game specified by I . Hence, we have established a polynomial-time reduction from TPCT to the complement of the QCP for Boolean mDatalog(τ_u) on unordered trees. This completes the proof of Proposition 6. \square

B.2 Omitting the predicates root and leaf: Proof of Theorem 1

From Proposition 6 we already know that the QCP is EXPTIME-hard for Boolean mDatalog($\tau_u^{\mathbf{root}, \mathbf{leaf}}$)-queries on unordered trees. Theorem 1 claims the same hardness result already for queries that don’t use the predicates **root** and **leaf**. Thus, Theorem 1 is an immediate consequence of Proposition 6 and the following lemma:

Lemma 8 *There is a polynomial-time reduction from the QCP for Boolean mDatalog($\tau_u^{\mathbf{root}, \mathbf{leaf}}$) on unordered trees to the QCP for Boolean mDatalog(τ_u) on unordered trees.*

Proof. Let Σ, Q_1, Q_2 be an input for the QCP for mDatalog($\tau_u^{\mathbf{root}, \mathbf{leaf}}$) on unordered trees. Our goal is to construct, within polynomial time, an alphabet $\tilde{\Sigma}$ and two Boolean mDatalog($\tau_{u, \tilde{\Sigma}}$)-queries \tilde{Q}_1, \tilde{Q}_2 , such that $Q_1 \subseteq Q_2$ iff $\tilde{Q}_1 \subseteq \tilde{Q}_2$.

We choose $\tilde{\Sigma} := \Sigma \times 2^{\{\mathbf{root}, \mathbf{leaf}\}}$. With every Σ -labeled tree T we associate the $\tilde{\Sigma}$ -labeled tree \tilde{T} that is obtained from T by replacing the label of each node $\alpha \in \Sigma$ with the label (α, I) where $I \subseteq \{\mathbf{root}, \mathbf{leaf}\}$ is given as follows:

$$\begin{aligned} \mathbf{root} \in I &\iff v \text{ is the root of } T, \\ \mathbf{leaf} \in I &\iff v \text{ is a leaf of } T. \end{aligned}$$

Let $\tilde{\mathcal{P}}_{labels}$ be the mDatalog($\tau_{u, \tilde{\Sigma}}$)-program consisting of the rules

$$\begin{aligned} \mathbf{label}_\alpha(x) &\leftarrow \mathbf{label}_{(\alpha, I)}(x) \\ \mathbf{root}(x) &\leftarrow \mathbf{label}_{(\alpha, I')}(x) \\ \mathbf{leaf}(x) &\leftarrow \mathbf{label}_{(\alpha, I'')}(x) \end{aligned}$$

for all $\alpha \in \Sigma$ and all $I, I', I'' \subseteq \{\mathbf{root}, \mathbf{leaf}\}$ with $\mathbf{root} \in I'$ and $\mathbf{leaf} \in I''$.

Let $\tilde{\mathcal{P}}_{incons}$ be the mDatalog($\tau_{u, \tilde{\Sigma}}$)-program consisting of the rules of $\tilde{\mathcal{P}}_{labels}$, along with the following rules:

$$\begin{aligned} P_{incons}(x) &\leftarrow \mathbf{root}(x), \mathbf{child}(y, x) \\ P_{incons}(x) &\leftarrow \mathbf{leaf}(x), \mathbf{child}(x, y) \\ P_{incons}(x) &\leftarrow \mathbf{child}(x, y), P_{incons}(y). \end{aligned}$$

The Boolean query $\tilde{Q}_{incons} = (\tilde{\mathcal{P}}_{incons}, P_{incons})$ describes all $\tilde{\Sigma}$ -labeled trees that are *inconsistent* in the sense that for any $\tilde{\Sigma}$ -labeled tree T' the following is true:

$$\tilde{Q}_{incons}(T') = \mathbf{yes} \iff \text{there is no } \Sigma\text{-labeled tree } T \text{ with } T' = \tilde{T}.$$

Now, for the given mDatalog($\tau_{b, \Sigma}$)-queries $Q_1 = (\mathcal{P}_1, P_1)$ and $Q_2 = (\mathcal{P}_2, P_2)$, we choose the mDatalog($\tau_{b, \tilde{\Sigma}}$)-queries $\tilde{Q}_1 = (\tilde{\mathcal{P}}_1, P_1)$ and $\tilde{Q}_2 = (\tilde{\mathcal{P}}_2, P_{acc})$ as follows:

$$\begin{aligned} \tilde{\mathcal{P}}_1 &:= \tilde{\mathcal{P}}_{labels} \cup \mathcal{P}_1, \\ \tilde{\mathcal{P}}_2 &:= \tilde{\mathcal{P}}_{incons} \cup \mathcal{P}_2 \cup \{ P_{acc}(x) \leftarrow P_{incons}(x), P_{acc}(x) \leftarrow P_2(x) \}. \end{aligned}$$

We claim that $Q_1 \not\subseteq Q_2 \iff \tilde{Q}_1 \not\subseteq \tilde{Q}_2$.

For the direction “ \implies ” let T be a Σ -labeled tree with $Q_1(T) = \mathbf{yes}$ and $Q_2(T) = \mathbf{no}$. Then, clearly, also $\tilde{Q}_1(\tilde{T}) = \mathbf{yes}$ and $\tilde{Q}_2(\tilde{T}) = \mathbf{no}$. Thus, $\tilde{Q}_1 \not\subseteq \tilde{Q}_2$.

For the direction “ \impliedby ” let T' be a Σ -labeled tree with $\tilde{Q}_1(T') = \mathbf{yes}$ and $\tilde{Q}_2(T') = \mathbf{no}$. The latter implies that T' is *not* inconsistent. Hence, there exists a Σ -labeled tree T such that $T' = \tilde{T}$. For this tree we know that $\tilde{Q}_1(\tilde{T}) = \mathbf{yes}$ and $\tilde{Q}_2(\tilde{T}) = \mathbf{no}$. Hence, also $Q_1(T) = \mathbf{yes}$ and $Q_2(T) = \mathbf{no}$. Thus, $Q_1 \not\subseteq Q_2$. This completes the proof of Lemma 8. \square

C EXPTIME-Hardness: Proof of Corollary 2

The aim of this appendix is to prove the following:

Corollary 2 (restated) *The QCP for Boolean mDatalog(τ_o)-queries on ordered trees is EXPTIME-hard.*

The proof is via a polynomial-time reduction from the QCP for Boolean mDatalog(τ_u)-queries over unordered trees which, according to Theorem 1, is EXPTIME-hard.

For establishing the reduction, we will rewrite monadic datalog programs of schema τ_u into suitable programs of schema τ_o (i.e., we will rewrite the **child** relation by means of the relations **fc** and **ns**). For doing this, we can use a result by Gottlob and Koch [11] which transforms monadic datalog programs into a certain normal form called *Tree-Marking Normal Form* (TMNF). We will use this normal form also later on, in Appendix D and Appendix E.

Definition 9 *Let τ be a schema that consists of relation symbols of arity at most 2. A monadic datalog program \mathcal{P} of schema τ is in TMNF if each rule of \mathcal{P} is of one of the following forms:³*

- $$\begin{array}{ll} (i) & X(x) \leftarrow R(x, y), Y(y) \\ (ii) & X(x) \leftarrow R(y, x), Y(y) \\ (iii) & X(x) \leftarrow Y(x), Z(x) \end{array}$$

where R is a binary predicate from τ , $X \in \text{idb}(\mathcal{P})$, and the unary predicates Y and Z are either intensional or belong to τ .

Theorem 10 (Gottlob and Koch [11, Theorem 5.2])

For each monadic datalog program \mathcal{P} of schema τ_{GK}^{child} , there is an equivalent program in TMNF of schema τ_{GK} , which can be computed in time $O(\|\mathcal{P}\|)$.

A detailed analysis shows that the proof given in [11] in fact also proves the following:

Corollary 11 (implicit in [11]) *For each monadic datalog program \mathcal{P} of schema τ_o^{child} , there is an equivalent program in TMNF of schema τ_o , which can be computed in time $O(\|\mathcal{P}\|)$.*

We are now ready for the proof of Corollary 2.

Proof of Corollary 2:

From Theorem 1 we already know the EXPTIME-hardness of the QCP for Boolean mDatalog(τ_u)-queries on unordered trees.

Thus, it suffices to give a polynomial-time reduction from this problem to the QCP for Boolean mDatalog(τ_o)-queries on ordered trees.

³ Gottlob and Koch [11] also allow rules of the form $X(x) \leftarrow Y(x)$. Note that such a rule is equivalent to the rule $X(x) \leftarrow Y(x), Y(x)$.

For this, note that $\tau_u \subseteq \tau_o^{\text{child}}$. Thus, upon input of two Boolean mDatalog(τ_u)-queries Q_1 and Q_2 , we can apply Corollary 11 to compute, in linear time, two Boolean mDatalog(τ_o)-queries Q'_1 and Q'_2 such that $Q'_i(T) = Q_i(T)$ is true for all *ordered* trees T and each $i \in \{1, 2\}$. Furthermore, since Q_i is of schema τ_u , we have that $Q_i(T) = Q_i(\tilde{T})$ is true for all ordered trees T and their unordered version \tilde{T} . Thus, we have $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$. I.e., we have established a polynomial-time reduction from the QCP for unordered trees to the QCP for ordered trees. This completes the proof of Corollary 2. \square

D EXPTIME-Membership: Proof of Theorem 3

The aim of this appendix is to prove the following Theorem:

Theorem 3 (restated) *The QCP for unary $\text{mDatalog}(\tau_{GK}^{\text{child}})$ -queries on ordered trees belongs to EXPTIME.*

We proceed as described in the proof sketch given in Section 3.

D.1 Step (1): From unary queries to Boolean queries

Let Σ be a finite alphabet, let T be an ordered Σ -labeled tree, and let v be a node of T . Considering the extended alphabet $\Sigma' := \Sigma \times \{0, 1\}$, we represent the tuple (T, v) by an ordered Σ' -labeled tree T'_v as follows: T'_v is obtained from T by changing the node labels, so that node v receives label $(\alpha_v, 1)$, and all further nodes u receive label $(\alpha_u, 0)$, where α_v and α_u denote the nodes' labels in T .

Lemma 12 *Every unary $\text{mDatalog}(\tau_{GK, \Sigma}^{\text{child}})$ -query Q can be rewritten, in linear time, into a Boolean $\text{mDatalog}(\tau_{GK, \Sigma'}^{\text{child}})$ -query Q'_{Bool} which satisfies the following:*

- For every ordered Σ -labeled tree T and every node v of T we have $v \in Q(T) \iff Q'_{\text{Bool}}(T'_v) = \text{yes}$.
- For every ordered Σ' -labeled tree T' with $Q'_{\text{Bool}}(T') = \text{yes}$, there are an ordered Σ -labeled tree T and a node v of T such that $T' = T'_v$.

Proof. Let $Q = (\mathcal{P}, P)$. We will construct Q'_{Bool} as follows:

- (i) Q'_{Bool} will simulate the program \mathcal{P} of Q .
- (ii) In parallel, Q'_{Bool} checks that the input tree contains exactly one node whose label is of the form $(\alpha, 1)$ for some $\alpha \in \Sigma$. We construct Q'_{Bool} in such a way that this is true iff the input tree's root node receives the intensional predicate C_1 .
- (iii) Finally, the root node receives the query predicate of Q'_{Bool} iff it has the C_1 -predicate and the query predicate P of the query Q contains a node of label $(\alpha, 1)$, for some $\alpha \in \Sigma$.

To this end, we let Q'_{Bool} be specified by a monadic datalog program \mathcal{P}' and a query predicate P' chosen as follows:

Start with $\mathcal{P}' := \emptyset$. For each letter $\alpha \in \Sigma$, we add to \mathcal{P}' the rules

$$\begin{array}{ll} \text{label}_\alpha(x) \leftarrow \text{label}_{(\alpha, 0)}(x) & X_0(x) \leftarrow \text{label}_{(\alpha, 0)}(x) \\ \text{label}_\alpha(x) \leftarrow \text{label}_{(\alpha, 1)}(x) & X_1(x) \leftarrow \text{label}_{(\alpha, 1)}(x) \end{array}$$

where X_0 and X_1 are unary relation symbols that do not occur in \mathcal{P} .

Next, add to \mathcal{P}' all rules of \mathcal{P} . Note that this way, we ensure that \mathcal{P}' simulates \mathcal{P} , and hence (i) is achieved.

To achieve (ii), we use two intensional predicates C_0, C_1 . We choose rules that proceed the binary tree built by the **fc** and **ns** relations in a bottom-up manner

and propagates, via the predicates C_0 and C_1 , whether the subtree rooted at the current node contains exactly 0 or exactly 1 nodes that carry the predicate X_1 . This is achieved by the following list of rules, which we add to \mathcal{P}' :

$$\begin{aligned}
C_0(x) &\leftarrow \mathbf{leaf}(x), \mathbf{ls}(x), X_0(x) \\
C_1(x) &\leftarrow \mathbf{leaf}(x), \mathbf{ls}(x), X_1(x) \\
\\
C_0(x) &\leftarrow \mathbf{leaf}(x), \mathbf{ns}(x, y), X_0(x), C_0(y) \\
C_1(x) &\leftarrow \mathbf{leaf}(x), \mathbf{ns}(x, y), X_0(x), C_1(y) \\
C_1(x) &\leftarrow \mathbf{leaf}(x), \mathbf{ns}(x, y), X_1(x), C_0(y) \\
\\
C_0(x) &\leftarrow \mathbf{ls}(x), \mathbf{fc}(x, y), X_0(x), C_0(y) \\
C_1(x) &\leftarrow \mathbf{ls}(x), \mathbf{fc}(x, y), X_0(x), C_1(y) \\
C_1(x) &\leftarrow \mathbf{ls}(x), \mathbf{fc}(x, y), X_1(x), C_0(y) \\
\\
C_0(x) &\leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_0(x), C_0(y), C_0(z) \\
C_1(x) &\leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_0(x), C_0(y), C_1(z) \\
C_1(x) &\leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_0(x), C_1(y), C_0(z) \\
C_1(x) &\leftarrow \mathbf{fc}(x, y), \mathbf{ns}(x, z), X_1(x), C_0(y), C_0(z)
\end{aligned}$$

Finally, we achieve (iii) by letting P' be a new intensional predicate and by adding to \mathcal{P}' the rule

$$P'(x) \leftarrow \mathbf{root}(x), C_1(x), P(y), X_1(y).$$

Clearly, \mathcal{P}' can be generated in time linear in the size of Q . □

As an immediate consequence, we obtain:

Lemma 13 *Let Σ be a finite alphabet and let $\Sigma' := \Sigma \times \{0, 1\}$. Within linear time, we can rewrite given unary $\mathbf{mDatalog}(\tau_{GK, \Sigma}^{\mathbf{child}})$ -queries Q_1 and Q_2 into Boolean $\mathbf{mDatalog}(\tau_{GK, \Sigma'}^{\mathbf{child}})$ -queries Q'_1 and Q'_2 such that $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$.*

Proof. For each $i \in \{1, 2\}$ let Q'_i be the query obtained by Lemma 12.

In case that $Q_1 \not\subseteq Q_2$, there are an ordered Σ -labeled tree T and a node v of T such that $v \in Q_1(T)$ and $v \notin Q_2(T)$. By Lemma 12 we obtain that $Q'_1(T'_v) = \mathbf{yes}$ and $Q'_2(T'_v) = \mathbf{no}$. Thus, $Q'_1 \not\subseteq Q'_2$.

In case that $Q'_1 \not\subseteq Q'_2$, there is an ordered Σ' -labeled tree T' such that $Q'_1(T') = \mathbf{yes}$ and $Q'_2(T') = \mathbf{no}$. Since $Q'_1(T') = \mathbf{yes}$, Lemma 12 tells us that there are an ordered Σ -labeled tree T and a node v of T such that $T' = T'_v$. Furthermore, by Lemma 12 we know that $v \in Q_1(T)$ and $v \notin Q_2(T)$. Thus, $Q_1 \not\subseteq Q_2$. □

Finally, we use Theorem 10 to eliminate the **child**-predicate and to obtain queries in TMNF.

Proposition 14 *Let Σ be a finite alphabet and let $\Sigma' := \Sigma \times \{0, 1\}$. Within linear time, we can rewrite given unary $\text{mDatalog}(\tau_{GK, \Sigma}^{\text{child}})$ -queries Q_1 and Q_2 into Boolean $\text{mDatalog}(\tau_{GK, \Sigma'})$ -queries Q'_1 and Q'_2 such that $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$. Furthermore, the programs of Q'_1 and Q'_2 are in TMNF.*

Proof. We apply Lemma 13 to obtain Boolean queries Q'_1 and Q'_2 . Afterwards, we apply Theorem 10 to rewrite the programs of the queries Q'_1 and Q'_2 into programs in TMNF of schema $\tau_{GK, \Sigma'}$. \square

Note that Proposition 14 partially establishes Step (1) of the agenda described in Section 3.

D.2 Step (1): From Ordered Unranked Trees to Binary Trees

For achieving Steps (2) and (3) we use, among other things, the classical notion of nondeterministic tree automata, which operate on ordered *binary* Σ -labeled trees. This subsection's goal is to fix notations concerning binary trees, and to show that, in order to prove Theorem 3, it suffices to find a 1-fold exponential algorithm that solves the QCP for Boolean queries in TMNF regarding binary trees.

Binary trees. An *ordered Σ -labeled binary tree* (for short: binary tree) $T = (V^T, \lambda^T, L^T, R^T)$ consists of a finite set V^T of nodes, a function $\lambda^T : V^T \rightarrow \Sigma$ assigning to each node v of T a label $\lambda^T(v) \in \Sigma$, and disjoint sets $L^T, R^T \subseteq V^T \times V^T$ such that the graph (V^T, E^T) with $E^T := L^T \cup R^T$ is a rooted directed tree where edges are directed from the root to the leaves, and each node has at most 2 children. For a tuple $(u, v) \in L^T$ (resp., R^T), we say that node v is the *left child* (resp., the *right child*) of node u .

We represent such a tree T as a relational structure of domain V^T with unary and binary relations: For each label $\alpha \in \Sigma$, $\text{label}_\alpha(x)$ expresses that x is a node with label α ; $\text{lc}(x, y)$ (resp., $\text{rc}(x, y)$) expresses that y is the left (resp., right) child of node x ; $\text{root}(x)$ expresses that x is the tree's root node; $\text{has_no_lc}(x)$ (resp., $\text{has_no_rc}(x)$) expresses that node x has no left child (resp., no right child), i.e., there is no node y with $(x, y) \in L^T$ (resp., R^T).

We denote this relational structure representing T by $\mathcal{S}_b(T)$, but when no confusion arises we simply write T instead of $\mathcal{S}_b(T)$. This relational structure is of schema

$$\tau_{b, \Sigma} := \{\text{lc}, \text{rc}\} \cup \{\text{root}, \text{has_no_lc}, \text{has_no_rc}\} \cup \{\text{label}_\alpha : \alpha \in \Sigma\}.$$

Representing Ordered Unranked Trees by Binary Trees. We use (a variant of) the standard representation (cf., e.g., [15]) of ordered unranked trees by binary trees. We represent an ordered Σ -labeled (unranked) tree T by a binary tree $\text{bin}(T)$ as follows: $\text{bin}(T)$ has the same vertex set and the same node labels as T , the “left child” relation $L^{\text{bin}(T)}$ consists of all tuples (x, y) such that

y is the first child of x in T (i.e., $\mathbf{fc}(x, y)$ is true in $\mathcal{S}_o(T)$), and the “right child” relation $R^{\mathbf{bin}(T)}$ consists of all tuples (x, y) such that y is the next sibling of x in T (i.e., $\mathbf{ns}(x, y)$ is true in $\mathcal{S}_o(T)$).

Note that the relational structure $\mathcal{S}_b(\mathbf{bin}(T))$ is obtained from the structure $\mathcal{S}_o(T)$ as follows:

- drop the relations **child** and **desc**,
- rename the relations **fc**, **ns**, **leaf**, **ls** into **lc**, **rc**, **has_no_lc**, **has_no_rc**, and
- insert the root node into the relation **has_no_rc**.

Furthermore, note that for a binary tree T' there exists an unranked ordered tree T with $T' = \mathbf{bin}(T)$ if, and only if, the root of T' has no right child (and in this case the tree T is unique).

Lemma 15. *Every Boolean mDatalog($\tau_{GK, \Sigma}$)-query Q can be rewritten, in linear time, into a Boolean mDatalog($\tau_{b, \Sigma}$)-query Q' which satisfies the following:*

- For every ordered Σ -labeled (unranked) tree T we have
 $Q(T) = \mathbf{yes} \iff Q'(\mathbf{bin}(T)) = \mathbf{yes}$.
- For every ordered Σ -labeled binary tree T' with $Q'(T') = \mathbf{yes}$ there is an ordered Σ -labeled (unranked) tree T such that $T' = \mathbf{bin}(T)$.

Furthermore, if the program of Q is in TMNF, then also the program of Q' is in TMNF.

Proof. Let $Q = (\mathcal{P}, P)$. We specify Q' by a monadic datalog program \mathcal{P}' and a query predicate P' as follows: \mathcal{P}' is obtained from \mathcal{P} by renaming, in each rule, the predicates **fc**, **ns**, **leaf**, **ls** into the predicates **lc**, **rc**, **has_no_lc**, **has_no_rc**. Furthermore, we let P' be a new intensional predicate, and we add to \mathcal{P}' the rule

$$P'(x) \leftarrow P(x), \mathbf{has_no_rc}(x).$$

It is straightforward to verify that the resulting Boolean query Q' has the desired properties. \square

By combining this lemma with Proposition 14, we obtain the following:

Proposition 16 *Let Σ be a finite alphabet and let $\Sigma' := \Sigma \times \{0, 1\}$. Within linear time, we can rewrite given unary mDatalog($\tau_{GK, \Sigma}^{\mathbf{child}}$)-queries Q_1 and Q_2 (querying ordered Σ -labeled unranked trees) into Boolean mDatalog($\tau_{b, \Sigma'}$)-queries Q'_1 and Q'_2 (querying ordered Σ' -labeled binary trees) such that $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$. Furthermore, the programs of Q'_1 and Q'_2 are in TMNF.*

Proof. We first apply Proposition 14 to obtain Boolean mDatalog($\tau_{GK, \Sigma'}$)-queries \tilde{Q}_1 and \tilde{Q}_2 , whose programs are in TMNF, such that $Q_1 \subseteq Q_2$ iff $\tilde{Q}_1 \subseteq \tilde{Q}_2$.

Next, we apply Lemma 15 to rewrite \tilde{Q}_1 and \tilde{Q}_2 into Boolean mDatalog($\tau_{b, \Sigma'}$)-queries Q'_1 and Q'_2 . It is straightforward to check that $\tilde{Q}_1 \subseteq \tilde{Q}_2$ iff $Q'_1 \subseteq Q'_2$:

In case that $\tilde{Q}_1 \not\subseteq \tilde{Q}_2$, there is an ordered Σ' -labeled unranked tree T such that $\tilde{Q}_1(T) = \mathbf{yes}$ and $\tilde{Q}_2(T) = \mathbf{no}$. By Lemma 15 we obtain that $Q'_1(\text{bin}(T)) = \mathbf{yes}$ and $Q'_2(\text{bin}(T)) = \mathbf{no}$. Thus, $Q'_1 \not\subseteq Q'_2$.

In case that $Q'_1 \not\subseteq Q'_2$, there is an ordered Σ' -labeled binary tree T' such that $Q'_1(T') = \mathbf{yes}$ and $Q'_2(T') = \mathbf{no}$. Since $Q'_1(T') = \mathbf{yes}$, Lemma 15 tells us that there is an ordered Σ' -labeled unranked tree T such that $T' = \text{bin}(T)$. Furthermore, by Lemma 15 we know that $\tilde{Q}_1(T) = \mathbf{yes}$ and $\tilde{Q}_2(T) = \mathbf{no}$. Thus, $\tilde{Q}_1 \not\subseteq \tilde{Q}_2$. \square

Proposition 16 implies that, in order to prove Theorem 3, it suffices to show that the following problem can be solved in 1-fold exponential time:

BOOLEAN-TMNF-QCP FOR MONADIC DATALOG ON BINARY TREES
Input: A finite alphabet Σ and two Boolean mDatalog(τ_b, Σ)-queries Q_1 and Q_2 whose programs are in TMNF.
Question: Is $Q_1 \subseteq Q_2$?

This finishes Step (1) of the agenda described in Section 3.

D.3 Step (2): Nondeterministic Bottom-Up Tree Automata (NBTA)

In this subsection we recall the classical notion (cf., e.g., [16]) of nondeterministic bottom-up tree automata (NBTA, for short), and show that a Boolean monadic datalog query Q on binary trees can be translated, within 1-fold exponential time, into an NBTA $A_Q^{\mathbf{no}}$ which accepts exactly those binary trees T for which $Q(T) = \mathbf{no}$.

A *nondeterministic bottom-up tree automaton* (NBTA, for short) A is specified by a tuple (Σ, S, Δ, F) , where Σ is a finite non-empty alphabet, S is a finite set of *states*, $F \subseteq S$ is the set of *accepting states*, and Δ is the *transition relation* with

$$\Delta \subseteq S_{\#} \times S_{\#} \times \Sigma \times S, \quad (1)$$

where $S_{\#} := S \cup \{\#\}$ for a symbol $\#$ that does not belong to S .

A *run* of A on an ordered Σ -labeled binary tree T is a mapping $\rho : V^T \rightarrow S$ such that the following is true for all nodes v of T , where α denotes the label of v in T :

- If v has no left child and no right child, then $(\#, \#, \alpha, \rho(v)) \in \Delta$.
- If v has a left child u_ℓ and a right child u_r , then $(\rho(u_\ell), \rho(u_r), \alpha, \rho(v)) \in \Delta$.
- If v has a left child u_ℓ , but no right child, then $(\rho(u_\ell), \#, \alpha, \rho(v)) \in \Delta$.
- If v has a right child u_r , but no left child, then $(\#, \rho(u_r), \alpha, \rho(v)) \in \Delta$.

A run ρ of A on T is *accepting* if $\rho(\text{root}^T) \in F$, where root^T is the root node of T . The automaton A *accepts* the tree T if there exists an accepting run of A on T . A tree T is *rejected* iff it is not accepted. The *tree language* $\mathcal{L}(A)$ is the set of all ordered Σ -labeled binary trees T that are accepted by A . A set L of ordered Σ -labeled binary trees is *regular* if $L = \mathcal{L}(A)$ for some NBTA A .

We define the *size* $\|\mathbf{A}\|$ of an NBTA \mathbf{A} to be the length of a reasonable representation of the tuple (Σ, S, Δ, F) ; to be precise, we let $\|\mathbf{A}\| := |\Sigma| + |S| + |\Delta| + |F|$. Note that due to (1) we have

$$\|\mathbf{A}\| = O(|S|^3 \cdot |\Sigma|). \quad (2)$$

It is well-known that the usual automata constructions for NFAs (i.e., non-deterministic finite automata on words) also apply to NBTAs. For formulating the results needed for our purposes, we introduce the following notation: For finite alphabets Σ and Γ we let proj_Σ be the mapping from $\Sigma \times \Gamma$ to Σ with $\text{proj}_\Sigma(\alpha, \beta) := \alpha$ for all $(\alpha, \beta) \in \Sigma \times \Gamma$. If T is a $(\Sigma \times \Gamma)$ -labeled tree, we write $\text{proj}_\Sigma(T)$ to denote the Σ -labeled tree obtained from T by replacing each node label (α, β) by the node label α .

By using standard automata constructions, one obtains:

Fact 17 (Folklore; see e.g. [5])

Union: For all NBTAs \mathbf{A}_1 and \mathbf{A}_2 over the same alphabet Σ , an NBTA \mathbf{A}_\cup with $\mathcal{L}(\mathbf{A}_\cup) = \mathcal{L}(\mathbf{A}_1) \cup \mathcal{L}(\mathbf{A}_2)$ can be constructed in time linear in $\|\mathbf{A}_1\|$ and $\|\mathbf{A}_2\|$. Furthermore, if k_i is the number of states of \mathbf{A}_i , for $i \in \{1, 2\}$, then the number of states of \mathbf{A}_\cup is $k_1 + k_2$.

Intersection: For all NBTAs \mathbf{A}_1 and \mathbf{A}_2 over the same alphabet Σ , an NBTA \mathbf{A}_\cap with $\mathcal{L}(\mathbf{A}_\cap) = \mathcal{L}(\mathbf{A}_1) \cap \mathcal{L}(\mathbf{A}_2)$ can be constructed in time polynomial in $\|\mathbf{A}_1\|$ and $\|\mathbf{A}_2\|$. Furthermore, if k_i is the number of states of \mathbf{A}_i , for $i \in \{1, 2\}$, then the number of states of \mathbf{A}_\cap is $k_1 \cdot k_2$.

Complementation: For every NBTA \mathbf{A} , an NBTA \mathbf{A}^c which accepts exactly those trees that are rejected by \mathbf{A} , can be constructed in time polynomial in $\|\mathbf{A}\| \cdot 2^k$, where k denotes the number of states of \mathbf{A} . Furthermore, the number of states of \mathbf{A}^c is 2^k .

Projection: For every NBTA \mathbf{A} over an alphabet of the form $\Sigma \times \Gamma$, an NBTA \mathbf{A}^p over alphabet Σ with $\mathcal{L}(\mathbf{A}^p) = \{\text{proj}_\Sigma(T) : T \in \mathcal{L}(\mathbf{A})\}$ can be constructed in time polynomial in $\|\mathbf{A}\|$. Furthermore, the number of states of \mathbf{A}^p is the same as the number of states of \mathbf{A} .

The emptiness problem for NBTAs is defined as follows:

EMPTINESS PROBLEM FOR NBTAs
Input: An NBTA $\mathbf{A} = (\Sigma, S, \Delta, F)$.
Question: Is $\mathcal{L}(\mathbf{A}) = \emptyset$?

Similarly as for NFAs, the emptiness problem for NBTAs can be solved efficiently:

Fact 18 (Folklore; see e.g. [5]) *The emptiness problem for NBTAs can be solved in time polynomial in the size of the input automaton.*

The following result establishes a relation between monadic datalog and NBTAs.

Proposition 19 *Let Σ be a finite alphabet and let Q be a Boolean $\text{mDatalog}(\tau_{b,\Sigma})$ -query whose program is in TMNF. Within time polynomial in $|\Sigma| \cdot 2^{\|Q\|}$ we can construct an NBTA \mathbf{A}^{no} with $2^{O(\|Q\|)}$ states, which accepts exactly those ordered Σ -labeled binary trees T where $Q(T) = \text{no}$.*

Proof. Our proof proceeds as described in the proof sketch given in Section 3. Let \mathcal{P} be the program of Q , let X_1 be the query predicate of Q , and let X_1, \dots, X_n be the list of all intensional predicates of \mathcal{P} .

Step 1: Transform Q into an equivalent monadic second-order sentence φ_Q :

We follow the “standard construction” (cf., [11, Proposition 3.3]), which uses the fact that the result $\mathcal{T}_{\mathcal{P}}^{\omega}(C)$ of a monadic datalog program \mathcal{P} on a set C of atomic facts is the *least fixed-point* of the immediate consequence operator $\mathcal{T}_{\mathcal{P}}$ that contains C :

For any rule r of \mathcal{P} of the form $h^r \leftarrow b_1^r, \dots, b_m^r$, define the formula

$$\psi_r := \forall z_1 \dots \forall z_\ell \left((b_1^r \wedge \dots \wedge b_m^r) \rightarrow h^r \right),$$

where z_1, \dots, z_ℓ is the list of variables appearing in the rule r . Since \mathcal{P} is in TMNF, we know that $m = 2$ and $\ell \leq 2$. W.l.o.g. we can assume that all rules use variables in $\{z_1, z_2\}$.

Let $\text{SAT}(X_1, \dots, X_n)$ be the conjunction of the formulas ψ_r for all rules r in \mathcal{P} , and let

$$\varphi_Q := \forall X_1 \dots \forall X_n \left(\text{SAT}(X_1, \dots, X_n) \rightarrow X_1(\text{root}) \right).$$

It is straightforward to verify (see [11, Proposition 3.3]) that for any ordered Σ -labeled binary tree T we have $Q(T) = \text{yes}$ if, and only if, the tree T , expanded by a constant *root* interpreted by the tree’s root node, satisfies the MSO-sentence φ_Q .

Clearly, φ_Q is equivalent to $\forall X_1 \dots \forall X_n \left(X_1(\text{root}) \vee \neg \text{SAT}(X_1, \dots, X_n) \right)$. Furthermore, $\neg \text{SAT}$ is equivalent to $\bigvee_{r \in \mathcal{P}} \neg \psi_r$; and $\neg \psi_r$ is equivalent to the formula $\exists z_1 \exists z_2 (b_1^r \wedge b_2^r \wedge \neg h^r)$, for a TMNF-rule r of the form $h^r \leftarrow b_1^r, b_2^r$. In summary, we obtain that φ_Q is equivalent to the formula

$$\varphi'_Q := \forall X_1 \dots \forall X_n \exists z_1 \exists z_2 \left(X_1(\text{root}) \vee \bigvee_{r \in \mathcal{P}} (b_1^r \wedge b_2^r \wedge \neg h^r) \right).$$

Clearly, for any tree T we have $Q(T) = \text{no}$ iff T satisfies the formula $\neg \varphi'_Q$, which is equivalent to the formula

$$\tilde{\varphi}_Q := \exists X_1 \dots \exists X_n \neg \exists z_1 \exists z_2 \left(X_1(\text{root}) \vee \bigvee_{r \in \mathcal{P}} (b_1^r \wedge b_2^r \wedge \neg h^r) \right).$$

Step 2: Transform $\tilde{\varphi}_Q$ into an equivalent NBTA:

We proceed in the same way as in well-known textbook proofs for Büchi’s Theorem, resp., the Theorem by Doner and Thatcher and Wright (stating the equivalence of MSO-definable languages and regular languages (of finite words and trees, respectively); cf. e.g. [16,8]):

Based on the formula $\tilde{\varphi}_Q$ we give the construction of the desired NBTA \mathbf{A}^{no} along the composition of the formula.

For the induction base, we have to handle quantifier-free formulas occurring in $\tilde{\varphi}_Q$. For this, we consider trees over alphabet $\Sigma_n := \Sigma \times \Gamma \times \Gamma'$ for $\Gamma := \{0, 1\}^n$ and $\Gamma' := \{0, 1\}^2$. If a node v has label $(\alpha, \gamma, \gamma')$, for $\gamma = \gamma_1 \cdots \gamma_n$ and $\gamma' = \gamma'_1 \gamma'_2$, we interpret this as the information that v has Σ -label α , belongs to the relation X_i iff $\gamma_i = 1$, and is the value of the variable z_j iff $\gamma'_j = 1$ (for $i \in \{1, \dots, n\}$ and $j \in \{1, 2\}$). We will refer to γ'_j (resp., γ_i and α) as the z_j -component (resp., the X_i -component and the Σ -component) of the label.

To check that the values in the z_j -components of a labeling indeed represent a variable assignment, we build for each $j \in \{1, 2\}$ an NBTA \mathbf{A}_{z_j} that accepts exactly those Σ_n -labeled trees where exactly one node carries a label whose z_j -component is 1. For example, the NBTA \mathbf{A}_{z_2} can be chosen as (Σ_n, S, Δ, F) with $S = \{s_0, s_1\}$, $F = \{s_1\}$, and Δ consisting of the transitions

$$(\#, \#, \beta, s_\nu), (s_0, s_0, \beta, s_\nu), (s_0, \#, \beta, s_\nu), (\#, s_0, \beta, s_\nu)$$

for all $\nu \in \{0, 1\}$ and all labels $\beta \in \Sigma \times \Gamma \times \{0, 1\} \times \{\nu\}$, and the transitions

$$(s_1, \#, \beta, s_1), (\#, s_1, \beta, s_1), (s_1, s_0, \beta, s_1), (s_0, s_1, \beta, s_1)$$

for all labels $\beta \in \Sigma \times \Gamma \times \{0, 1\} \times \{0\}$. This automaton performs a bottom-up scan of the tree and remains in state s_0 until it encounters a node whose label has a 1 in its z_2 -component. The latter induces a change into state s_1 . The automaton gets stuck (i.e., no run exists) if it is in state s_1 and encounters another node whose label has a 1 in its z_2 -component.

To check whether an atomic or negated atomic formula χ (occurring in $\tilde{\varphi}_Q$) is satisfied by an input tree, we build an NBTA \mathbf{A}_χ that accepts an input tree T iff T contains, for each variable z_j occurring in χ , a node v_j whose z_j -component is 1, such that the nodes v_j satisfy χ . If χ involves a *unary* atom, this can be achieved in a straightforward way using an automaton with 2 states. If χ is a *binary* atom, this is not difficult either. E.g., if $\chi = \mathbf{lc}(z_2, z_1)$, the NBTA \mathbf{A}_χ performs a bottom-up scan of the tree and remains in state s_0 until it encounters a node v_1 whose z_1 -component is labeled 1. The latter induces a change into state s_1 . From there on, the automaton either gets stuck, or it sees that v_1 is the left child of a node v_2 whose z_2 -component is one. The latter induces a change into an accepting state s_2 , which is propagated to the root.

Note that each of the NBTAs constructed so far has at most 3 states and, according to (2), size $O(3^3 \cdot |\Sigma_n|) = O(|\Sigma_n|)$.

The formula $\tilde{\varphi}_Q$ contains a conjunction ζ_r of the form $(b_1^r \wedge b_2^r \wedge \neg h^r)$, for each rule $r \in \mathcal{P}$. We already have available NBTAs $\mathbf{A}_{b_1^r}$, $\mathbf{A}_{b_2^r}$, $\mathbf{A}_{\neg h^r}$, \mathbf{A}_{z_1} , \mathbf{A}_{z_2} , each of which has at most 3 states and size $O(|\Sigma_n|)$. By using the intersection-construction mentioned in Fact 17, we can build the intersection automaton \mathbf{A}_{ζ_r} of these five NBTAs. This can be achieved in time polynomial in $O(|\Sigma_n|)$; and the resulting automaton has at most 3^5 states and thus, due to (2), size $O(|\Sigma_n|)$.

The quantifier-free part of the formula $\tilde{\varphi}_Q$ is the disjunction of the formula $X_1(\text{root})$ and the formulas ζ_r , for all $r \in \mathcal{P}$. We already have available NBTAs $\mathbf{A}_{X_1(\text{root})}$ and \mathbf{A}_{ζ_r} for each $r \in \mathcal{P}$. Using the union-construction mentioned in

Fact 17, we can build the union automaton A_{gf} of these automata. This can be achieved in time polynomial in $O(|\mathcal{P}| \cdot |\Sigma_n|)$; and the resulting automaton has at most $(|\mathcal{P}|+1) \cdot 3^5 = O(|\mathcal{P}|)$ states and thus, due to (2), size $O(|\mathcal{P}|^3 \cdot |\Sigma_n|)$.

Note that A_{gf} is an NBTA over alphabet $\Sigma \times \Gamma \times \Gamma'$. We now use the projection-construction mentioned in Fact 17 to build an NBTA $A_{\exists z_1 \exists z_2}$ accepting the set of all trees of the form $proj_{\Sigma \times \Gamma}(T)$, for T accepted by A_{gf} . The resulting automaton has the same number of states as A_{gf} , i.e., $O(|\mathcal{P}|)$, has size $O(|\mathcal{P}|^3 \cdot |\Sigma \times \Gamma|) = O(|\mathcal{P}|^3 \cdot |\Sigma| \cdot 2^n)$, and can be constructed in time polynomial in $O(|\mathcal{P}|^3 \cdot |\Sigma_n|)$.

Next, we use the complementation-construction mentioned in Fact 17 to build an NBTA A_{\neg} which accepts exactly those trees that are rejected by $A_{\exists z_1 \exists z_2}$. The automaton A_{\neg} has $2^{O(|\mathcal{P}|)}$ states and thus, due to (2), size $O(2^{O(|\mathcal{P}|)} \cdot |\Sigma \times \Gamma|) = O(2^{O(|\mathcal{P}|)} \cdot |\Sigma| \cdot 2^n)$. It can be constructed in time polynomial in the size of $A_{\exists z_1 \exists z_2}$ and $2^{O(|\mathcal{P}|)}$, i.e., polynomial in $2^{O(|\mathcal{P}|)} \cdot |\mathcal{P}|^3 \cdot |\Sigma| \cdot 2^n$.

Finally, we use the projection-construction mentioned in Fact 17 to build an NBTA $A^{\mathbf{no}}$ accepting the set of all trees of the form $proj_{\Sigma}(T)$, for T accepted by A_{\neg} . The resulting automaton has the same number of states as A_{\neg} , i.e., $2^{O(|\mathcal{P}|)}$ and can be constructed in time polynomial in the size of A_{\neg} , i.e., polynomial in $2^{O(|\mathcal{P}|)} \cdot |\Sigma| \cdot 2^n = |\Sigma| \cdot 2^{n+O(|\mathcal{P}|)} = |\Sigma| \cdot 2^{O(\|Q\|)}$.

It is straightforward to verify that the NBTA $A^{\mathbf{no}}$ accepts exactly those Σ -labeled trees T that satisfy the formula $\tilde{\varphi}_Q$, i.e., those trees T with $Q(T) = \mathbf{no}$. The entire construction of the automaton $A^{\mathbf{no}}$ took time polynomial in $|\Sigma| \cdot 2^{\|Q\|}$. This completes the proof of Proposition 19. \square

This establishes the “ $A^{\mathbf{no}}$ -part” of Step (2) of the agenda described in Section 3. By applying to $A^{\mathbf{no}}$ the complementation-construction mentioned in Fact 17, we obtain an NBTA $A^{\mathbf{yes}}$ which accepts exactly the Σ -labeled trees T with $Q(T) = \mathbf{yes}$. However, the number of states of $A^{\mathbf{no}}$ is $2^{O(\|Q\|)}$, and hence the construction of $A^{\mathbf{yes}}$ takes time polynomial in $\|A^{\mathbf{no}}\| \cdot 2^{2^{O(\|Q\|)}}$, which is 2-fold exponential in the size of the query Q .

To construct an NBTA equivalent to $A^{\mathbf{yes}}$ within 1-fold exponential time, we use a different automata model, described in the next subsection.

D.4 Step (2): 2-way alternating tree automata (2ATA)

In this subsection we recall the notion (cf., e.g., [6,17,14]) of 2-way alternating tree automata (2ATA), and show that a Boolean monadic datalog query Q on binary trees can be translated, within polynomial time, into a 2ATA $\hat{A}^{\mathbf{yes}}$ which accepts exactly those binary trees T for which $Q(T) = \mathbf{yes}$. The following definitions concerning 2ATAs are basically taken from [6,17].

For navigating in a binary tree T we consider the operations *up*, *stay*, *left*, *right*. They are viewed as functions from V_{\perp}^T to V_{\perp}^T where $V_{\perp}^T = V^T \cup \{\perp\}$ for the node set V^T of T and a symbol \perp not in V^T . Each of the operations in $Op := \{up, stay, left, right\}$ maps \perp to \perp . Furthermore, for each node v of T , we have $stay(v) = v$, while $up(v)$ is the parent of v in T (resp. \perp , in case that v is

the root of T), and $left(v)$ is the left child of v in T (resp. \perp , in case that v has no left child), and $right(v)$ is the right child of v in T (resp. \perp , in case that v has no right child).

Let M be a set. The set $\mathcal{B}^+(M)$ of *positive Boolean formulas over M* contains all elements in M , and is closed under \wedge and \vee . For a set $M' \subseteq M$ and a formula $\theta \in \mathcal{B}^+(M)$, we say that M' *satisfies* θ iff assigning **true** to elements in M' and **false** to elements in $M \setminus M'$ makes θ true.

A *two-way alternating tree automaton* (2ATA, for short) \hat{A} is specified by a tuple $(\Sigma, S, s_0, \delta, F)$, where

- Σ is a finite non-empty alphabet,
- S is a finite set of states,
- $s_0 \in S$ is the *initial* state,
- $F \subseteq S$ is the set of *accepting* states, and
- $\delta : S \times \Sigma \rightarrow \mathcal{B}^+(S \times Op)$ is the *transition function*.

As input, \hat{A} receives a Σ -labeled binary tree T . It starts in the initial state s_0 at T 's root node. Whenever \hat{A} is in a state $s \in S$ and currently visits a node v of T of label $\alpha \in \Sigma$, it can either choose to stop its computation, or to perform a further step in which the formula $\theta := \delta(s, \alpha)$ determines what is done next: the automaton nondeterministically guesses a satisfying assignment for θ , i.e., a set $\{(s_1, o_1), \dots, (s_k, o_k)\}$ (for some $k \geq 1$) which satisfies θ . Then, it starts k independent copies of \hat{A} , namely a copy which starts in state s_i at node $o_i(v)$, for each $i \in \{1, \dots, k\}$. In case that $o_i(v) = \perp$, the according automaton stops. The acceptance condition demands that for every situation (s, v) in which the automaton stops, s must be an accepting state.

This can be formalised by the following notion of a *run* R , where the label (s, o, v) of a node w of R denotes a transition into state s via the operation o onto node v .

A *run* of \hat{A} on a Σ -labeled binary tree T is a finite unordered *unranked* Γ -labeled tree R , for $\Gamma := S \times Op \times V_\perp^T$, which satisfies the following conditions:

- (1) The root of R is labeled with $(s_0, stay, root^T)$, where s_0 is the initial state and $root^T$ is the root of T .
- (2) If w is a node of R that is labeled (s, o, v) with $v = \perp$, then w is a *leaf* of R .
- (3) If w is a node of R that is labeled (s, o, v) such that v is a node of T , and w' is a child of w in R that is labeled (s', o', v') , then $v' = o'(v)$.
- (4) If w is a node of R that is labeled (s, o, v) such that v is a node of T labeled $\alpha \in \Sigma$, and w has exactly k children labeled $(s_1, o_1, v_1), \dots, (s_k, o_k, v_k)$, then the formula $\theta := \delta(s, \alpha)$ is satisfied by the set $\{(s_1, o_1), \dots, (s_k, o_k)\}$.

A run R of \hat{A} on T is *accepting* if every leaf of R is labeled with an accepting state, i.e.: whenever (s, o, v) is the label of a leaf of R , we have $s \in F$. The automaton \hat{A} *accepts* the tree T if there exists an accepting run of \hat{A} on T . The *tree language* $\mathcal{L}(\hat{A})$ is the set of all ordered Σ -labeled binary trees T that are accepted by \hat{A} .

The *size* $\|\hat{A}\|$ of a 2ATA \hat{A} is defined as the length of a reasonable representation of the tuple $(\Sigma, S, s_0, \delta, F)$.

It is known that 2ATAs accept exactly the same tree languages as NBTAs, i.e., the *regular* tree languages. Furthermore, there is a 1-fold exponential algorithm that translates a 2ATA into an equivalent NBTA:

Theorem 20 (Cosmadakis et al. [6]) *For every 2ATA \hat{A} , an NBTA A with $\mathcal{L}(A) = \mathcal{L}(\hat{A})$ can be constructed within time 1-fold exponential in $\|\hat{A}\|$.*

To be precise, [6] formulated the theorem not in terms of the running time, but only in terms of the *size* of the generated NBTA. A proof sketch of the theorem can be found in [6]; detailed proofs of more general results can be found in [17,14].

Our next goal is to find a polynomial-time algorithm which translates a Boolean monadic datalog query Q in TMNF into an equivalent 2ATA \hat{A} which accepts exactly those trees T with $Q(T) = \mathbf{yes}$.

To construct such a 2ATA, we will exploit the striking similarity between runs of 2ATAs and *proof trees* characterising the semantics of datalog (cf., the textbook [2]). For constructing the desired 2ATA, the following observation will be very convenient:

Let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF, and let \mathcal{P} and P be the program and the query predicate of Q . For a Σ -labeled binary tree T with root node $root^T$ we have $Q(T) = \mathbf{yes}$ iff there exists a proof tree PT for the fact $P(root^T)$, such that the leaves of the proof tree are labeled with facts in $atoms(\mathcal{S}_b(T))$. Note that, for the particular case of TMNF-programs, such a proof tree PT has the following properties:

- The *root* of PT is labeled with the atomic fact $P(root^T)$.
- Each *leaf* of PT is labeled with an atomic fact of one of the following forms:
 - **label** $_{\alpha}(v)$ where $\alpha \in \Sigma$ and v is a node of T labeled α ,
 - **root** $(root^T)$, where $root^T$ is the root of T ,
 - **has_no_lc** (v) (resp., **has_no_rc** (v)), where v is a node of T that has no left child (resp., has no right child)
 - **lc** (v_1, v_2) (resp., **rc** (v_1, v_2)), where v_2 is the left (resp., right) child of v_1 in T .
- Each non-leaf node of PT is labeled with a fact $X(v)$ where v is a node of T and $X \in \text{idb}(\mathcal{P})$.
- Every non-leaf node w of PT has exactly 2 children w_1 and w_2 . If w is labeled by an atomic fact $X(v)$, then \mathcal{P} contains a rule r whose head is of the form $X(x)$, and the following is true:
 - (a) If the body of r is of the form $Y(x), Z(x)$, then w_1 is labeled $Y(v)$ and w_2 is labeled $Z(v)$.
 - (b) If the body of r is of the form **lc** $(x, y), Y(y)$ then node v of T has a left child v' , and in PT the nodes w_1 and w_2 are labeled with the facts **lc** (v, v') and $Y(v')$.
Accordingly, if the body of r is of the form **rc** $(x, y), Y(y)$ then node v of T has a right child v' , and in PT the nodes w_1 and w_2 are labeled with the facts **rc** (v, v') and $Y(v')$.

- (c) If the body of r is of the form $\mathbf{lc}(y, x), Y(y)$, then node v of T is the left child of its parent v' , and in PT the nodes w_1 and w_2 are labeled with the facts $\mathbf{lc}(v', v)$ and $Y(v')$.
 Accordingly, if the body of r is of the form $\mathbf{rc}(y, x), Y(y)$, then node v of T is the right child of its parent v' , and in PT the nodes w_1 and w_2 are labeled with the facts $\mathbf{rc}(v', v)$ and $Y(v')$.

We will build a 2ATA for which an accepting run R on an input tree T precisely corresponds to a proof tree PT for the fact $P(\text{root}^T)$. To better cope with technical details in the automaton construction, we will consider automata which receive input trees that are labeled by the extended alphabet $\hat{\Sigma}$, with

$$\hat{\Sigma} := \Sigma \times 2^{\{\mathbf{root}, \mathbf{has_no_lc}, \mathbf{has_no_rc}, \mathbf{is_lc}, \mathbf{is_rc}\}}.$$

With every Σ -labeled binary tree T we associate a $\hat{\Sigma}$ -labeled binary tree \hat{T} that is obtained from T by replacing the label of each node v labeled $\alpha \in \Sigma$ with the label (α, I) where $I \subseteq \{\mathbf{root}, \mathbf{has_no_lc}, \mathbf{has_no_rc}, \mathbf{is_lc}, \mathbf{is_rc}\}$ is given as follows:

$$\begin{aligned} \mathbf{root} \in I &\iff v \text{ is the root of } T, \\ \mathbf{has_no_lc} \in I &\iff v \text{ is a node of } T \text{ that has no left child,} \\ \mathbf{has_no_rc} \in I &\iff v \text{ is a node of } T \text{ that has no right child,} \\ \mathbf{is_lc} \in I &\iff v \text{ is the left child of its parent } v' \text{ in } T, \\ \mathbf{is_rc} \in I &\iff v \text{ is the right child of its parent } v' \text{ in } T. \end{aligned}$$

We are now ready for this subsection's key result:

Proposition 21 *Let Σ be a finite alphabet and let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF. Within time polynomial in the size of Q and Σ , we can construct a 2ATA \hat{A} such that for all Σ -labeled binary trees T , the automaton \hat{A} accepts the tree \hat{T} if, and only if, $Q(T) = \text{yes}$.*

Proof. Let \mathcal{P} and P be the program and the query predicate of Q . We construct the automaton \hat{A} in such a way that a proof tree PT for the fact $P(\text{root}^T)$ can be easily be turned into an accepting run of \hat{A} on \hat{T} (and vice versa).

The state set S of the $\hat{A} = (\hat{\Sigma}, S, s_0, \delta, F)$ is chosen as the set all intensional predicates of \mathcal{P} , all unary relation symbols in $\tau_{b,\Sigma}$, and additionally, we use states called **is_lc**, **is_rc**, **accept**, and **reject**. I.e.,

$$\begin{aligned} S = & \{\mathbf{accept}, \mathbf{reject}\} \cup \text{idb}(\mathcal{P}) \cup \\ & \{\mathbf{label}_\alpha : \alpha \in \Sigma\} \cup \{\mathbf{root}, \mathbf{has_no_lc}, \mathbf{has_no_rc}, \mathbf{is_lc}, \mathbf{is_rc}\}. \end{aligned}$$

The query predicate P is the initial state, and **accept** is the only accepting state. I.e., $s_0 := P$ and $F := \{\mathbf{accept}\}$.

The transition function $\delta : S \times \hat{\Sigma} \rightarrow \mathcal{B}^+(S \times Op)$ is chosen as follows:

Let $\beta = (\alpha, I)$ be an arbitrary letter in $\hat{\Sigma}$. We let

$$\delta(\mathbf{accept}, \beta) := (\mathbf{accept}, \text{stay}) \quad \text{und} \quad \delta(\mathbf{reject}, \beta) := (\mathbf{reject}, \text{stay}).$$

For every $\alpha' \in \Sigma$ we let

$$\delta(\text{label}_{\alpha'}, \beta) := \begin{cases} (\text{accept}, \text{stay}) & \text{if } \alpha' = \alpha \\ (\text{reject}, \text{stay}) & \text{otherwise.} \end{cases}$$

For every $X \in \{\text{root}, \text{has_no_lc}, \text{has_no_rc}, \text{is_lc}, \text{is_rc}\}$ we let

$$\delta(X, \beta) := \begin{cases} (\text{accept}, \text{stay}) & \text{if } X \in I \\ (\text{reject}, \text{stay}) & \text{otherwise.} \end{cases}$$

For the case that $X \in \text{idb}(\mathcal{P})$, the formula $\delta(X, \beta)$ is specified as follows. We let \mathcal{P}_X be the set of all rules of \mathcal{P} whose head is of the form $X(x)$, and we choose

$$\delta(X, \beta) := \bigvee_{r \in \mathcal{P}_X} \theta_r,$$

where the formula $\theta_r \in \mathcal{B}^+(S \times Op)$ is chosen as indicated in the following table:

rule r of the form	conjunction θ_r
$X(x) \leftarrow Y(x), Z(x)$	$(Y, \text{stay}) \wedge (Z, \text{stay})$
$X(x) \leftarrow \text{lc}(x, y), Y(y)$	$(\text{is_lc}, \text{left}) \wedge (Y, \text{left})$
$X(x) \leftarrow \text{rc}(x, y), Y(y)$	$(\text{is_rc}, \text{right}) \wedge (Y, \text{right})$
$X(x) \leftarrow \text{lc}(y, x), Y(y)$	$(\text{is_lc}, \text{stay}) \wedge (Y, \text{up})$
$X(x) \leftarrow \text{rc}(y, x), Y(y)$	$(\text{is_rc}, \text{stay}) \wedge (Y, \text{up})$

Clearly, this automaton $\hat{\mathbf{A}}$ can be constructed in time polynomial in the size of Σ and Q . It remains to verify that, indeed, for any Σ -labeled binary tree T we have $Q(T) = \text{yes} \iff \hat{\mathbf{A}}$ accepts \hat{T} .

For the “ \implies ”-direction, let PT be a proof tree for the the fact $P(\text{root}^T)$. We can transform PT into a run R of $\hat{\mathbf{A}}$ on \hat{T} as follows: Assign the new label $(P, \text{stay}, \text{root}^T)$ to the root node of PT . For each non-leaf node w of PT note that w is originally labeled by an atomic fact $X(v)$ with $X \in \text{idb}(\mathcal{P})$, and w has exactly two children w_1, w_2 in PT .

- (a) If w_1, w_2 are labeled $Y(v), Z(v)$, then assign to node w_1 the new label (Y, stay, v) and to node w_2 the new label (Z, stay, v) .
- (b) If w_1, w_2 are labeled $\text{lc}(v, v'), Y(v')$, then assign to node w_1 the new label $(\text{is_lc}, \text{left}, v')$ and to node w_2 the new label (Y, left, v') . Furthermore, we add to w_1 a new child labeled $(\text{accept}, \text{stay}, v)$.
We proceed analogously in case that w_1, w_2 is labeled $\text{rc}(v, v'), Y(v')$.
- (c) If w_1, w_2 are labeled $\text{lc}(v', v), Y(v')$, then assign to w_1 the new label $(\text{is_lc}, \text{stay}, v)$, and to node w_2 the new label (Y, up, v') . Furthermore, we add to w_1 a new child labeled $(\text{accept}, \text{stay}, v)$.
We proceed analogously in case that w_1, w_2 is labeled $\text{rc}(v', v), Y(v')$.

Finally, for each leaf w of PT that was originally labeled $X(v)$ for an $X \in \{\mathbf{root}, \mathbf{has_no_lc}, \mathbf{has_no_rc}\} \cup \{\mathbf{label}_\alpha : \alpha \in \Sigma\}$, we add a new child w_1 that receives the new label $(\mathbf{accept}, \mathbf{stay}, v)$.

It is straightforward to verify that the obtained tree R is an accepting run of \hat{A} on \hat{T} .

For the direction “ \Leftarrow ” let R be an accepting run of \hat{A} on \hat{T} . Along the definition of δ it is straightforward to see that we can assume w.l.o.g. that each node of R has at most 2 children.

The run R can be turned into a proof tree PT for the fact $P(\mathbf{root}^T)$ (i.e., witnessing that $Q(T) = \mathbf{yes}$) as follows: Consider each node w of R , and let (s, o, v) be the label of node w .

Since R is an *accepting* run and \mathbf{accept} is the only accepting state, we know by the construction of δ that $s \neq \mathbf{reject}$, and that $v \neq \perp$ if $s \neq \mathbf{accept}$. In case that $s \in \tau_{b,\Sigma} \cup \text{idb}(\mathcal{P})$, we assign to w the new label “ $s(v)$ ”.

In case that $s = \mathbf{label}_{\alpha'}$ for an $\alpha' \in \Sigma$, we know by the construction of δ and the fact that R is an *accepting* run, that node w has a unique child w_1 in R , and this node w_1 is labeled with $(\mathbf{accept}, \mathbf{stay}, v)$. Furthermore, we know by the construction of δ that $\alpha' = \alpha$ where $\beta = (\alpha, I)$ is the label of node v in \hat{T} . Thus, the statement “ $\mathbf{label}_{\alpha'}(v)$ ” is true for node v in T . Hence, we delete the node w_1 (and all nodes in the subtree rooted at w_1).

In case that $s \in \{\mathbf{root}, \mathbf{has_no_lc}, \mathbf{has_no_rc}, \mathbf{is_lc}, \mathbf{is_rc}\}$, we know by the construction of δ and the fact that R is an *accepting* run, that node w has a unique child w_1 in R , and this node w_1 is labeled with $(\mathbf{accept}, \mathbf{stay}, v)$. Furthermore, we know by the construction of δ that $s \in I$, where $\beta = (\alpha, I)$ is the label of node v in \hat{T} . Thus, the statement “ $s(v)$ ” is true for node v in T . Hence, we delete the node w_1 (and all nodes in the subtree rooted at w_1).

In case that $s \in \{\mathbf{root}, \mathbf{has_no_lc}, \mathbf{has_no_rc}\}$, the node w then is a leaf, labeled with an atomic fact “ $s(v)$ ” that is true in T .

In case that $s = \mathbf{is_lc}$, the statement “ $\mathbf{is_lc}(v)$ ” is a true statement, but it is not suitable as label in a proof tree, since the predicate $\mathbf{is_lc}$ does not belong to the schema $\tau_{b,\Sigma}$. Therefore, we replace the label “ $\mathbf{is_lc}(v)$ ” by the label “ $\mathbf{lc}(v', v)$ ” where v' is the parent of v in T . We proceed analogously in case that $s = \mathbf{is_rc}$.

It is straightforward to verify that the obtained tree PT is a proof tree for $P(\mathbf{root}^T)$. This completes the proof of Proposition 21. \square

Finally, we are ready for establishing the second part of Step 2 of the agenda described in Section 3.

Proposition 22 *Let Σ be a finite alphabet and let Q be a Boolean mDatalog($\tau_{b,\Sigma}$)-query whose program is in TMNF. Within time 1-fold exponential in the size of Q and Σ , we can construct an NBTA $A^{\mathbf{yes}}$, which accepts exactly those ordered Σ -labeled binary trees T where $Q(T) = \mathbf{yes}$.*

Proof. First, we use Proposition 21 to construct, within polynomial time, a 2ATA \hat{A} such that for all ordered binary Σ -labeled trees T , the automaton \hat{A} accepts the $\hat{\Sigma}$ -labeled tree \hat{T} if, and only if, $Q(T) = \mathbf{yes}$.

Now, we use Theorem 20 to construct, within time 1-fold exponential in $\|\hat{\mathbf{A}}\|$ (i.e., 1-fold exponential in the size of Q and Σ), an NBTA \mathbf{A} with $\mathcal{L}(\mathbf{A}) = \mathcal{L}(\hat{\mathbf{A}})$.

Note that \mathbf{A} operates on $\hat{\Sigma}$ -labeled trees, while we are looking for an NBTA \mathbf{A}^{yes} operating on Σ -labeled trees. To obtain such an automaton, we proceed as follows:

Let \mathbf{B} be an NBTA of alphabet $\hat{\Sigma}$ which accepts exactly those $\hat{\Sigma}$ -labeled trees T' for which there exists a Σ -labeled tree T such that $T' = \hat{T}$ (building such an NBTA is straightforward: the automaton just needs to check that the $\hat{\Sigma}$ -labels correctly identify the root node, the nodes that are left (right) children, and the nodes that have no left (right) child).

Using the intersection-construction mentioned in Fact 17, we can build the intersection automaton \mathbf{A}' of \mathbf{B} and \mathbf{A} . I.e., \mathbf{A}' accepts a $\hat{\Sigma}$ -labeled tree T' iff there exists a Σ -labeled tree T such that $T' = \hat{T}$, and \hat{T} is accepted by \mathbf{A} .

Finally, we use the projection-construction described in Fact 17 to obtain an NBTA \mathbf{A}^{yes} over alphabet Σ , such that $\mathcal{L}(\mathbf{A}^{\text{yes}}) = \{\text{proj}_{\Sigma}(T') : T' \in \mathcal{L}(\mathbf{A}')\}$. Thus, \mathbf{A}^{yes} accepts a tree $T \iff \hat{T}$ is accepted by $\mathbf{A} \iff Q(T) = \text{yes}$.

Since the intersection- and projection-constructions can be performed within time polynomial in the size of its input NBTAs, the entire construction of \mathbf{A}^{yes} takes time at most 1-fold exponential in the size of Q and Σ . \square

D.5 Step (3): Finishing the proof of Theorem 3

Proof of Theorem 3:

Our goal is to show that the QCP for unary mDatalog(τ_{GK}^{child})-queries on ordered trees belongs to EXPTIME.

Let Σ, Q_1, Q_2 be an input for the QCP. Let $\Sigma' := \Sigma \times \{0, 1\}$. By using Proposition 16 we obtain, within linear time, Boolean mDatalog($\tau_{b, \Sigma'}$)-queries Q'_1 and Q'_2 such that $Q_1 \subseteq Q_2$ iff $Q'_1 \subseteq Q'_2$, and the programs of Q'_1 and Q'_2 are in TMNF.

By using Proposition 22, we can construct, within time 1-fold exponential in the size of Q'_1 and Σ' , an NBTA $\mathbf{A}_1^{\text{yes}}$, which accepts exactly those Σ' -labeled binary trees T where $Q'_1(T) = \text{yes}$.

By using Proposition 19, we can construct, within time 1-fold exponential in the size of Q'_2 and Σ' , an NBTA \mathbf{A}_2^{no} , which accepts exactly those Σ' -labeled binary trees T where $Q'_2(T) = \text{no}$.

Now, we use the intersection-construction mentioned in Fact 17 to build the intersection-automaton \mathbf{B} of $\mathbf{A}_1^{\text{yes}}$ and \mathbf{A}_2^{no} . Clearly, \mathbf{B} accepts a Σ' -labeled binary tree T if, and only if, $Q'_1(T) = \text{yes}$ and $Q'_2(T) = \text{no}$.

Finally, we use the emptiness-test provided by Fact 18 to check whether $\mathcal{L}(\mathbf{B}) = \emptyset$. Clearly, this is the case if, and only if, $Q'_1 \subseteq Q'_2$, which in turn is true iff $Q_1 \subseteq Q_2$.

Since the intersection-construction and the emptiness test take only time polynomial in the size of the input automata, the entire algorithm for checking whether $Q_1 \subseteq Q_2$ runs in time 1-fold exponential in the size of Σ, Q_1 , and Q_2 . This completes the proof of Theorem 3 \square

E Dealing with the descendant-axis: Proof of Theorem 5

The aim of this appendix is to prove the following:

Theorem 5 (restated) *The QCP for unary $\text{mDatalog}(\tau_u^{\text{root,leaf,desc}})$ on unordered trees and for unary $\text{mDatalog}(\tau_{GK}^{\text{child,desc}})$ can be solved in 2-fold exponential time.*

Note that $\tau_u^{\text{root,leaf,desc}} \subseteq \tau_{GK}^{\text{child,desc}}$. Thus, to prove Theorem 5, it suffices to provide a 2-fold exponential algorithm for the QCP for unary $\text{mDatalog}(\tau_{GK}^{\text{child,desc}})$ -queries on ordered trees.

Upon input of two $\text{mDatalog}(\tau_{GK}^{\text{child,desc}})$ -queries Q_1 and Q_2 , our algorithm proceeds as follows: First, we transform Q_1 and Q_2 into equivalent queries Q'_1 and Q'_2 that do *not* contain the **desc**-predicate. Afterwards, we use the algorithm provided by Theorem 3 to decide whether $Q'_1 \subseteq Q'_2$. Thus, Theorem 5 is an immediate consequence of Theorem 3 and the following Lemma 23.

Lemma 23 *For every $\text{mDatalog}(\tau_{GK}^{\text{child,desc}})$ -query Q there is an equivalent $\text{mDatalog}(\tau_{GK}^{\text{child}})$ -query Q' , which can be computed in 1-fold exponential time.*

The remainder of Appendix E is devoted to the proof of Lemma 23.

The proof proceeds in three steps:

Step 1: Gottlob, Koch, and Schulz [12, Theorem 6.6] showed that every conjunctive query using the axes **child**, **desc**, **ns** can be rewritten, in 1-fold exponential time, into an equivalent union of *acyclic* conjunctive queries. We extend their result to monadic datalog rules that may also contain the **fc**-relation (see Lemma 26 below).

Step 2: Afterwards, we use a result of [11] which shows that every *acyclic* conjunctive query can be rewritten, in linear time, into a monadic datalog program that is “almost” in TMNF (see Lemma 25 below).

Step 3: Finally, observe that each TMNF-rule which uses the **desc**-relation can be replaced (in constant time) by two suitable rules using **child** (see Fact 24).

Step 3 is established by the following obvious fact:

Fact 24

Over trees, the rule $X(x) \leftarrow \text{desc}(x, y), Y(y)$ is equivalent to the rules

$$\begin{aligned} X(x) &\leftarrow \text{child}(x, y), Y(y) \\ X(x) &\leftarrow \text{child}(x, y), X(y). \end{aligned}$$

Similarly, the rule $X(x) \leftarrow \text{desc}(y, x), Y(y)$ is equivalent to the rules

$$\begin{aligned} X(x) &\leftarrow \text{child}(y, x), Y(y) \\ X(x) &\leftarrow \text{child}(y, x), X(y). \end{aligned}$$

For Steps 1 and 2, let us recall the notion of *acyclic* queries considered in [11,12]. Let τ be a schema consisting of relations of arity at most 2. Let r be a rule of a monadic datalog query of schema τ . The *directed rule graph* G_r is the multigraph whose vertex set is the set of variables of r , and where for each binary atom of the form $R(x, y)$ occurring in the rule's body, there is a directed edge e_R from node x to node y . The *shadow* of G_r is the undirected multigraph obtained from G_r by ignoring the edge directions. We say that r contains a *directed cycle* if the multigraph G_r contains a directed cycle. Accordingly, r contains an *undirected cycle* if the shadow of G_r contains a cycle. A rule is called *acyclic* if it does not contain an undirected cycle; an mDatalog(τ)-program is *acyclic* if all its rules are acyclic.

Step 2 of our agenda is provided by the following lemma.

Lemma 25 ([11, Lemma 5.8]) *Let r be an acyclic monadic datalog rule over relations that are either unary or binary. Then, r can be decomposed in linear time into a monadic datalog program in which each rule is one of the three forms*

$$X(x) \leftarrow R(y, x), Y(y) \quad X(x) \leftarrow R(x, y), Y(y) \quad X(x) \leftarrow Y(x), Z(z)$$

where x (resp., Y) may but does not have to be different from z (resp., Z).

Finally, Step 1 of our agenda is established by the following Lemma 26, which generalises a result by Gottlob, Koch, and Schulz [12, Theorem 6.6] to queries that may make use of the **fc**-predicate.

Lemma 26 *Every unary mDatalog($\tau_{GK}^{\text{child, desc}}$)-query Q can be rewritten, in 1-fold exponential time, into an equivalent mDatalog($\tau_{GK}^{\text{child, desc}}$)-query Q' such that each rule in the program of Q' is acyclic.*

Proof. Let \mathcal{P} be the program of Q . We choose Q' to have the same query predicate as Q . The program \mathcal{P}' of Q' is constructed as follows.

We initialise \mathcal{P}' to be equal to \mathcal{P} . Then, while \mathcal{P}' is *not* acyclic, do the following: Let r be a rule in \mathcal{P}' that is not acyclic. Remove r from \mathcal{P}' .

Case 1: If r contains a directed cycle, note that r is not satisfiable (since the directed cycle is built from the axes **fc**, **ns**, **child**, **desc**).

Thus, we simply drop r .

Case 2: Otherwise, r must contain an undirected cycle, but no directed cycle. Then, the directed query graph G_r is a DAG, and there must exist a variable z of r which belongs to an undirected cycle, such that G_r contains no directed path from z to another variable that belongs to an undirected cycle. For this variable z , the rule's body must contain two atoms of the form $R(x, z)$ and $S(y, z)$ (where $R, S \in \{\mathbf{fc}, \mathbf{ns}, \mathbf{child}, \mathbf{desc}\}$, and x, y are variables). We make the following case distinction:

- (i) In case that $R = \mathbf{fc}$ and $S = \mathbf{ns}$ (or vice versa), note that the rule is unsatisfiable, and hence we simply drop r .

- (ii) In case that $R = S \in \{\mathbf{fc}, \mathbf{ns}, \mathbf{child}\}$, note that $R(x, z) \wedge S(y, z)$ is equivalent to $R(x, z) \wedge y=x$. Thus, we let \tilde{r} be the rule obtained from r by omitting the atom $S(y, z)$ and replacing all occurrences of y by x . We add \tilde{r} to \mathcal{P}' .
- (iii) In case that $R = S = \mathbf{desc}$, note that $R(x, z) \wedge S(y, z)$ is equivalent to $\varphi := (\mathbf{desc}(x, y) \wedge \mathbf{desc}(y, z)) \vee (\mathbf{desc}(y, x) \wedge \mathbf{desc}(x, z)) \vee (\mathbf{desc}(x, z) \wedge y=z)$.

For each $i \in \{1, 2, 3\}$ we let \tilde{r}_i be the rule obtained from r by replacing “ $R(x, z), S(y, z)$ ” with the i -th clause of φ . Concerning \tilde{r}_3 , we furthermore delete the atom $y=z$ and replace all occurrences of y by z . We add \tilde{r}_1, \tilde{r}_2 , and \tilde{r}_3 to \mathcal{P}' .

- (iv) In case that $R = \mathbf{fc}$ and $S = \mathbf{child}$ (or vice versa), note that $R(x, z) \wedge S(y, z)$ is equivalent to $R(x, z) \wedge y=z$. Hence, we proceed in the same way as in case (ii).
- (v) In case that $R = \mathbf{ns}$ and $S \in \{\mathbf{child}, \mathbf{desc}\}$ (or vice versa), note that $R(x, z) \wedge S(y, z)$ is equivalent to $R(x, z) \wedge S(y, x)$. We let \tilde{r} be the rule obtained from r by replacing the atom $S(y, z)$ with the atom $S(y, x)$, and we add \tilde{r} to \mathcal{P}' .
- (vi) In case that $R \in \{\mathbf{fc}, \mathbf{child}\}$ and $S = \mathbf{desc}$ (or vice versa), note that $R(x, z) \wedge S(y, z)$ is equivalent to $\varphi :=$

$$(R(x, z) \wedge \mathbf{desc}(y, x)) \vee (R(x, z) \wedge y=z).$$

For each $i \in \{1, 2\}$ we let \tilde{r}_i be the rule obtained from r by replacing “ $R(x, z), S(y, z)$ ” with the i -th clause of φ . Concerning \tilde{r}_2 , we furthermore delete the atom $y=z$ and replace all occurrences of y by z . We add \tilde{r}_1 and \tilde{r}_2 to \mathcal{P}' .

Clearly, the obtained query Q' is equivalent to the original query Q . Furthermore, along the same lines as in the proof of [12, Lemma 6.4], one can show that the algorithm terminates after a number of steps that is at most 1-fold exponential in the size of the input query Q . Of course, upon termination the program \mathcal{P}' is acyclic. Thus, the proof of Lemma 26 is complete. \square

Finally, we are ready for the proof of Lemma 23.

Proof of Lemma 23:

Let Q be the given $\text{mDatalog}(\tau_{GK}^{\mathbf{child}, \mathbf{desc}})$ -query.

Using Lemma 26 we construct, within 1-fold exponential time, an equivalent $\text{mDatalog}(\tau_{GK}^{\mathbf{child}, \mathbf{desc}})$ -query Q_1 such that each rule in the program \mathcal{P}_1 of Q_1 is acyclic. By applying Lemma 25 to each rule of \mathcal{P}_1 , we obtain an equivalent $\text{mDatalog}(\tau_{GK}^{\mathbf{child}, \mathbf{desc}})$ -query Q_2 such that each rule in the program \mathcal{P}_2 of Q_2 is one of the following forms:

$$X(x) \leftarrow R(y, x), Y(y) \quad X(x) \leftarrow R(x, y), Y(y) \quad X(x) \leftarrow Y(x), Z(z)$$

with $R \in \{\mathbf{fc}, \mathbf{ns}, \mathbf{child}, \mathbf{desc}\}$.

Applying Fact 24, we then replace every rule of \mathcal{P}_2 that contains the **desc**-relation by two rules that use the **child**-relation.

This leads to an mDatalog($\tau_{GK}^{\mathbf{child}}$)-query Q_3 that is equivalent to Q . Furthermore, Q_3 is computed in time 1-fold exponential in the size of Q . \square